Titre: Title:	Intégration d'un modèle de réseau sur puce dans un flôt de conception de niveau système
Auteur: Author:	Hubert Guérard
Date:	2011
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Guérard, H. (2011). Intégration d'un modèle de réseau sur puce dans un flôt de conception de niveau système [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/751/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/751/
Directeurs de recherche: Advisors:	Guy Bois
Programme: Program:	Génie informatique

UNIVERSITÉ DE MONTRÉAL

INTÉGRATION D'UN MODÈLE DE RÉSEAU SUR PUCE DANS UN FLÔT DE CONCEPTION DE NIVEAU SYSTÈME

HUBERT GUÉRARD

DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE INFORMATIQUE)

DÉCEMBRE 2011

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

INTÉGRATION D'UN MODÈLE DE RÉSEAU SUR PUCE DANS UN FLÔT DE CONCEPTION DE NIVEAU SYSTÈME

présenté par : GUÉRARD Hubert

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. BELTRAME, Giovanni, Ph.D., président

M. BOIS, Guy, Ph.D., membre et directeur de recherche

M. DAVID, Jean-Pierre, Ph.D., membre

REMERCIEMENTS

Je tiens à remercier mon directeur de recherche, le professeur Guy Bois, pour son soutien, ses conseils et également pour m'avoir initié à la recherche.

Au sein du groupe, je tiens à remercier plusieurs collègues qui m'ont aidé de près ou de loin, soit Luc Filion, Sébastien Le Beux et Laurent Moss.

De plus, je ne peux pas passer sous silence les heures de plaisirs que j'ai passé avec mes nombreux collègues de travail soit, Jean-François Cartier, Gabriel Cartier, Michel Rogers-Vallée, Mathieu McKinnon, Jessica Allard-Bernier, Fellipe Montero, Vincent Legault et Gabriel Oshiro. Sans eux, mon expérience aurait été différente.

Finalement, je tiens à remercier ma copine, Anny Sylvestre ainsi que ma mère Pierrette Audesse pour leur support moral, leur compréhension ainsi que leur encouragement tout au long de mon projet.

RÉSUMÉ

Les systèmes embarqués deviennent plus complexes puisqu'ils incluent beaucoup de ressources et doivent réaliser plusieurs fonctionnalités. Ceci introduit un problème au niveau de l'interconnexion des ressources, car un grand volume de données doit être traité. Une solution proposée est l'utilisation de réseau sur puce (abrégé par l'acronyme NoC signifiant Network-on-chip). Par ailleurs, la conception et la simulation d'une architecture écrite dans un langage de description matériel nécessite beaucoup d'effort. Ceci est attribuable à la granularité fine de tels langages. Une alternative est le recours à une méthodologie haut niveau (ESL) qui propose l'utilisation de modèles abstraits (abstraction des communications et des calculs) afin de simuler et valider le système plus rapidement et plus tôt.

L'outil haut niveau nommé SPACE consiste à une plate-forme virtuelle permettant la simulation, la validation, l'exploration architecturale et l'implémentation de ces mêmes architectures en utilisant les topologies de communication traditionnelle. Puisque le réseau sur puce est un concept émergeant, SPACE ne supporte pas l'utilisation de réseaux sur puce. Dans cette optique, l'objectif de ce travail consiste à étendre la librairie de topologie de communication de SPACE jusqu'aux réseaux sur puce tout en préservant ses fonctionnalités de base.

À terme, l'intégration d'un réseau sur puce dans l'outil SPACE permettra d'élargir l'espace de solution à explorer, de faire du co-design logiciel/matériel, d'obtenir des métriques de performances du système et même d'obtenir une implémentation bas niveau pouvant s'exécuter sur une puce FPGA.

ABSTRACT

Nowadays, embedded systems become more complex as they integrate more resources performing several functions. The interconnection of many resources introduces a communication problem since a large volume of data must be processed. A solution is the use of a network-on-chip (NoC). Furthermore, the design and simulation of an architecture written in a hardware description language requires a lot of effort. This is due to the granularity of such languages. An alternative is to use a high-level methodology (Electronic System Level methodology), which use abstract models (higher abstraction communications and calculations) to achieve faster simulation and hence, faster system deployment

The high-level tool named SPACE is a virtual platform for simulation, validation, architectural exploration and implementation of architectures using traditional communication. Since the network-on-chip is an emerging concept, SPACE does not support the use of network-on-chip. The objective of this work is to extend the communication library to network-on-chip while preserving the basic functionality of SPACE.

At the end of this project, the integration of a network-on-chip in SPACE will expand the solution space, allow co-design of hardware/software, obtain performance metrics and even to obtain a low-level implementation that can run on a FPGA chip.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

AMBA	Advanced Microcontroller Bus Architecture
API	Application Programmable Interface
BCA	Bus Cycle Accurate
EDA	Electronic Design Automation
EIB	Element Interconnect Bus
ESL	Electronic system-level
FIFO	First-In First-Out
FPGA	Field-programmable gate array
FSL	Fast Simplex Link
JPEG	Joint Photographic Experts Group
HAL	Hardware Abstraction Layer
HDL	Hardware description language
ISS	Instruction Set Simulator
MJPEG	Motion JPEG
MPSoC	Multi-processors System On-Chip
NI	Network Interface
NOC	Network-on-chip
OCP	Open Core Protocol
OS	Operating System
PA-BCA	Pin-Accurate Bus Cycle Accurate
ROC	Rotator-on-chip
RTL	Register-transfer level
RTOS	Real-Time Operating System

SOC	System-on-chip
T-BCA	Transaction based Bus Cycle Accurate
TF	Timed Functional
TLM	Transaction-level-modeling
UTF	Untimed functional
VCI	Virtual Component Interface

CHAPITRE 1 INTRODUCTION

1.1 Mise en contexte et problématique

De nos jours, il est presque inimaginable de ne pas consommer des biens électroniques puisque le monde qui nous entoure devient de plus en plus informatisé. La consommation de tels produits est donc une pratique très répandue. Il suffit de penser au « Black Friday » aux États-Unis où l'on s'arrache le dernier modèle de télévision à rabais. Un autre bel exemple de consommation de biens électroniques est le fameux « iPhone » d'Apple vendu à travers le monde. Qu'il s'agisse de télévisions ou de téléphones cellulaires, tous ces appareils ont tous un point en commun : les systèmes embarqués.

Un système embarqué est un système servant à résoudre des fonctions et des tâches spécifiques et limitées [46]. Ces tâches peuvent être simples ou complexes en fonction de l'objectif à atteindre. Ceci étant dit, les systèmes embarqués sont très présents tout autour de nous sans même s'en rendre compte. Afin de répondre à la demande grandissante des consommateurs, les concepteurs de produits doivent créer des systèmes plus élaborés ce qui nécessite de créer des systèmes embarqués plus complexes.

Jadis considéré comme étant un facteur limitant dans la conception de systèmes embarqués, le nombre de transistors que l'on peut mettre sur une puce de silicium ne cesse d'augmenter. En effet, selon la loi de Moore [1], le nombre de transistors que l'on peut mettre sur une même puce double chaque deux ans. Ceci a pour effet de repousser les limites des systèmes embarqués. Il devient alors possible de concevoir des systèmes plus complexes. Par exemple, il est maintenant imaginable de concevoir un système embarqué capable de traiter un flot vidéo en haute définition en temps réel, ce qui n'était pas le cas il y a quelques années.

Cependant, la conception d'une telle application crée un problème au niveau des communications, car un grand volume de données doit être traité. Il n'est pas possible d'utiliser les topologies de communications traditionnelles, car elles ne sont pas adaptées pour supporter

l'interconnexion de plusieurs ressources. Il faut plutôt utiliser le concept de topologie de bus avancé tel que le réseau sur puce (Network-on-chip). Ce concept de topologie a comme caractéristique de pouvoir interconnecter plusieurs ressources sans pour autant dégrader les communications.

Le récent intérêt apporté par la communauté scientifique au concept de réseau sur puce dans les systèmes embarqués a permis la création de diverses topologies avancées. Parmi ces topologies, notons-le « rotator-on-chip » [3] qui représente une topologie en anneau semblable au « token ring » [4] dans les réseaux informatiques. Une topologie de communication similaire est le « Element Interconnect Bus » (EIB) [5] développé par le consortium formé par Sony, Toshiba et IBM.

Bien que l'utilisation de réseaux sur puce soit une réalité industrielle, l'implémentation et la simulation de ses topologies écrites dans un langage de description matériel tel que VHDL [6] ou Verilog [7] nécessitent beaucoup d'efforts [8]. Par conséquent, la validation d'un système à concevoir utilisant un réseau sur puce peut s'avérer très fastidieux dû à la complexité de la topologie. Ceci est attribuable à la granularité fine de ces langages.

Une méthodologie de simulation à haut niveau peut remédier à ce problème [2]. Cette méthodologie, appelée « Electronic System Level » (ESL) [9], permet d'abstraire des détails d'implémentation qui ne sont pas pertinents lors d'une simulation à haut niveau. En effectuant cette abstraction, la simulation du système à concevoir s'effectue plus rapidement et plus facilement sans pour autant perdre de la précision [10]. Les concepteurs de systèmes ont avantage à tirer profit d'une telle approche, car elle permet de valider rapidement les spécifications initiales.

Par exemple, dans une abstraction de type T-BCA « Transaction Bus Cycle Accurate » [9], le modèle haut niveau conserve les latences responsables de la précision du modèle au niveau du minutage (en anglais *timing*). En d'autres termes, lorsqu'on désire effectuer une simulation

haut niveau, on abstrait les détails d'implémentation, on garde seulement l'essentiel du modèle en minimisant les altérations du comportement final du système. Puisque le comportement haut niveau est semblable au comportement bas niveau correspondant, il est dans notre intérêt d'utiliser une méthode de conception à haut niveau, car il est plus facile d'y œuvrer grâce à l'abstraction.

Dans une telle approche, au lieu d'utiliser un langage de description matériel, on utilise le langage de programmation C/C++ ainsi qu'une librairie de simulation haut niveau tel que SystemC [11] afin de créer les modèles. L'intégration des réseaux sur puce dans les outils ESL est une approche récente. Ceci est dû à l'émergence des réseaux sur puce.

1.2 Objectifs

L'objectif principal de ce travail est d'intégrer un réseau sur puce dans un outil haut niveau afin de permettre la simulation et la validation. Cet objectif se décompose en trois objectifs spécifiques.

1.2.1 Objectifs spécifiques

1) Étude des réseaux sur puce et de la méthodologie ESL

Établir l'état de l'art dans le domaine des réseaux sur puce et du concept de l'ESL.

2) Élaboration et intégration des modèles

Concevoir un modèle haut et bas niveau d'un réseau sur puce. Intégrer ces modèles dans l'outil SPACE.

3) Évaluation des modèles

Simulation et validation des modèles à l'aide des métriques développées et de banc d'essais. Une implémentation sur puce FPGA permettra de valider l'implémentation ainsi que le temps d'exécution.

1.3 Méthodologie

Le projet se divise en quatre phases, soit la phase d'élaboration, la phase de développement, la phase d'évaluation et finalement, la phase de raffinement.

1.3.1 Phase I : Élaboration

La phase d'élaboration consiste à recueillir les informations nécessaires au projet grâce à la revue de la littérature. Elle comporte les activités suivantes :

Détermination des requis d'un réseau sur puce

Grâce à une revue de la littérature et à une analyse approfondie des réseaux sur puce, cette activité consiste à expliquer qu'est-ce qu'un réseau sur puce, les différentes méthodologies de conception ainsi que les caractéristiques que doivent posséder ces topologies qui pourraient éventuellement être utilisées dans une intégration dans un outil à haut niveau. Par exemple, les réseaux sur puce peuvent se caractériser en termes de latence, nombre de nœuds, etc. Ces informations sont essentielles lors de la réalisation d'un modèle haut niveau. De plus, cette revue littéraire permettra de déterminer quel modèle de réseau sur puce sera retenu lors de la phase de développement. Cette activité est donc préliminaire à l'atteinte du premier objectif spécifique.

Étude de la simulation haut niveau

Une fois les caractéristiques d'un réseau sur puce déterminées, il faut connaitre les techniques de conceptualisation de modèles haus niveau. Pour ce faire, la revue de la littérature effectuée au point précédent couvera également le domaine de la simulation haut niveau. Par exemple, il faut identifier une libraire de simulation ainsi que les distinctions entre les différents niveaux d'abstraction. Cette activité permettra d'atteindre le premier objectif spécifique.

1.3.2 Phase II : Développement

La phase de développement consiste à développer les outils nécessaires à l'évaluation des réseaux sur puce. Toutes les activités de cette phase visent à atteindre le second objectif spécifique. Lesdites activités sont les suivantes :

Création des modèles

La première étape de la phase de développement consiste à créer un modèle haut niveau et bas niveau d'un réseau sur puce. L'élaboration du modèle haut niveau sera faite à l'aide de la librairie de simulation. De plus, l'intégration des métriques de performances sera utile lors de la dernière phase. L'élaboration du modèle synthétisable bas niveau sera faite à l'aide du langage VHDL.

Intégration dans l'outil SPACE

La seconde activité de cette phase consiste à intégrer les modèles dans l'outil haut niveau tout en préservant les fonctionnalités. Le modèle haut niveau sera utilisé lors des simulations tandis que le modèle bas niveau sera utile lors du raffinement de l'architecture.

1.3.3 Phase III : Évaluation

Cette phase consiste à effectuer la comparaison des modèles (haut niveau et bas niveau) et elle comporte deux activités qui visent à atteindre le troisième objectif spécifique.

Évaluation du modèle haut niveau

Cette activité consiste à s'assurer que le modèle haut niveau se trouve dans le niveau d'abstraction désirée. Les métriques de performances développées lors de la phase de développement permettront de valider le niveau d'abstraction.

Comparaison des modèles

Cette activité consiste à simuler le modèle haut niveau développé lors de la deuxième phase. À l'aide des résultats obtenus grâce aux métriques de performances développées, des résultats seront produits dans le but de comparer le comportement du modèle haut niveau à celui du modèle bas niveau. Nous utiliserons les résultats publiés comme référence pour le modèle bas niveau. Le comportement du modèle haut niveau sera validé si ce dernier est comparable à celui du modèle bas niveau.

1.3.4 Raffinement

Cette étape consiste à la dernière phase de ce projet. Cette phase consiste à effectuer une implémentation sur puce FPGA d'une architecture test intégrant le modèle bas niveau suivant la méthodologie proposée par l'outil SPACE.

1.3.4.1 Implémentation sur cible FPGA

Valider le fonctionnement d'une architecture intégrant un réseau sur puce. Comparez le temps simulé au temps exécuté. Au terme de cette activité, l'objectif du projet sera atteint.

Notez que M. Gabriel Oshiro contribuera au développement du raffinement d'une architecture vers une cible FPGA dans le cadre de son mémoire [12].

1.4 Contributions

Dans le cadre de ce travail, le réseau sur puce intégré dans l'outil SPACE ne permettra pas uniquement de simuler un réseau sur puce à haut niveau. Il tira avantage de la plate-forme SPACE. Ceci comprend la possibilité de faire du co-design (matériel/logiciel), de faire l'exploration architecturale, déplacer les tâches logicielles vers le matériel et vice-versa facilement, d'obtenir des métriques de performances du système de façon non intrusif. Mais avant tout, la principale contribution est qu'il sera possible, à partir même de cette plate-forme virtuelle, de générer une implémentation matérielle et d'exécuter cette dernière sur une puce FPGA.

Ce mémoire est organisé comme suit. Le deuxième chapitre présente la revue de littérature qui traitera dans un premier temps des réseaux sur puce et puis du domaine de l'ESL. Le troisième chapitre développe notre méthodologie et le quatrième chapitre expose les résultats obtenus. Finalement, le dernier chapitre effectue une synthèse des travaux suivis d'une description des améliorations futures.

CHAPITRE 2 REVUE DE LITTÉRATURE

Ce chapitre présente les travaux pertinents dans le cadre de ce mémoire. Plus particulièrement, nous avons besoin de présenter deux domaines de recherche. Le premier domaine présenté est celui des réseaux sur puce et le second est le domaine de l'ESL.

On passe d'abord en revue les travaux portant sur le domaine des réseaux sur puce en y présentant une explication du concept de réseau sur puce. Par la suite, nous traiterons des méthodologies de conception et finalement des architectures de réseau sur puce.

Suivra par la suite la revue du domaine de l'ESL en discutant premièrement des différents niveaux d'abstractions, des diverses stratégies de conception d'adaptateur et finalement, nous traiterons de divers outils utilisant le concept de l'ESL.

2.1 Réseau sur puce

D'amblé de jeu, un réseau sur puce est une topologie de communication à l'instar d'un bus traditionnel. Cela étant dit, la tâche première d'un NoC est d'échanger des informations d'un point vers un autre point en offrant de meilleures performances que le bus, et ce non seulement au niveau de la bande passante, mais aussi du point de vue évolutif, tolérance au pannes, etc.

Dans [15], les auteurs dessinent le portrait global du domaine des réseaux sur puce en divisant le domaine de recherche en sous-domaine. Par la suite, chaque sous-domaine est traité de façon unique. Cet article est en quelque sort le point de départ de tout néophyte du domaine. Ici, la partie intéressante de cet article est la façon dont les concepts de base sont expliqués.

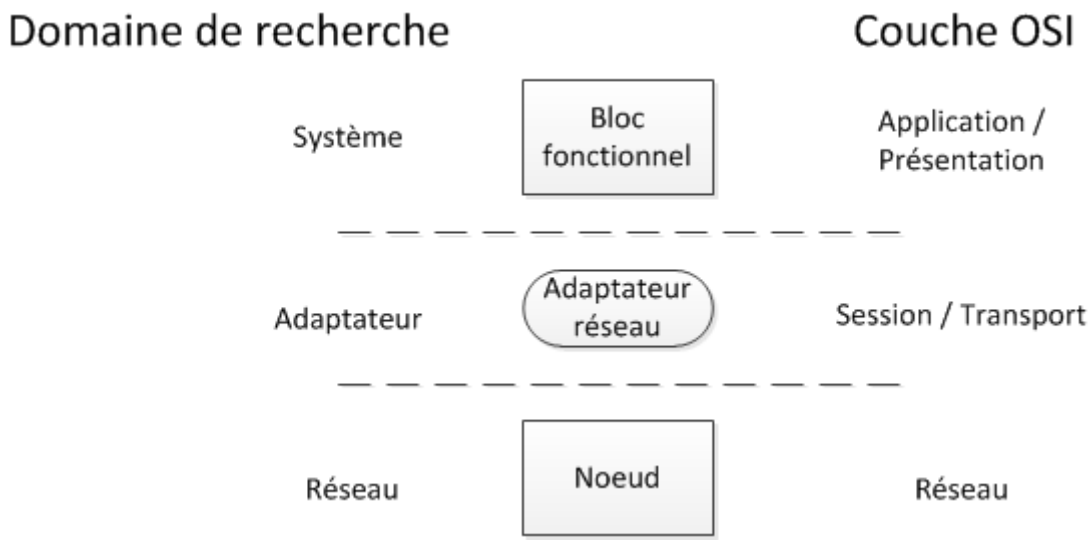

Figure 2-1 : Modèle OSI et réseau sur puce

Un réseau sur puce est une topologie de communication qui applique le concept des réseaux informatiques (protocole avancé, topologie de communication, communication par paquet, etc.) [13]. Le modèle OSI [19] sépare les communications au travers d'un réseau en sept couches distinctes. Chaque couche est alors responsable d'effectuer une tâche précise. Les réseaux sur puce intègrent le concept de fragmentation des tâches. La Figure 2-1, tiré de [15], montre une correspondance entre les diverses couches du modèle OSI avec les éléments d'un réseau sur puce.

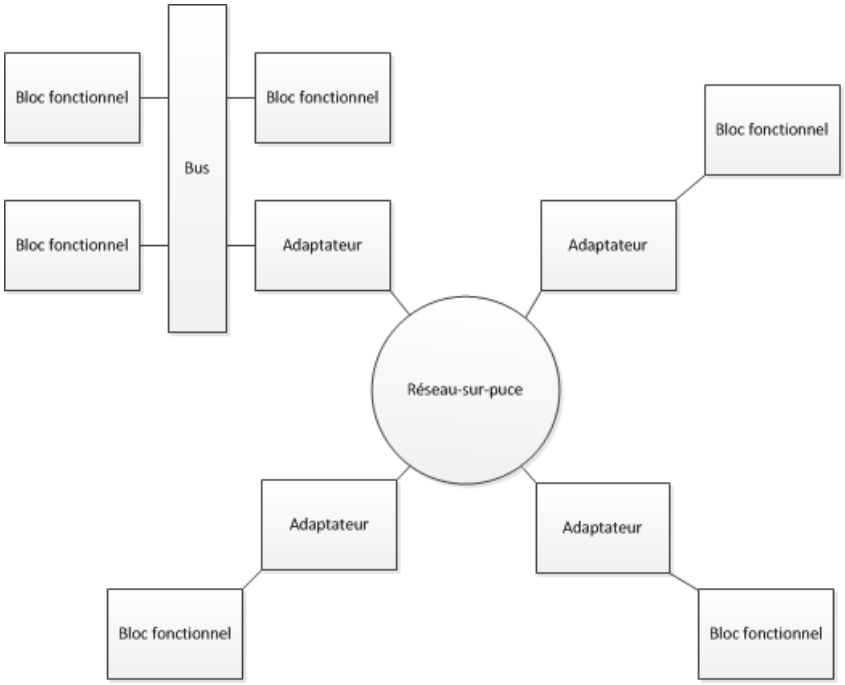

Figure 2-2: Exemple de réseau sur puce

Un nœud désigne ce qui est branché au NoC. Un nœud est constitué d'un adaptateur et d'un bloc fonctionnel ou d'un ensemble de blocs fonctionnels. Lorsqu'on parle d'un ensemble de blocs fonctionnels, on utilise le terme « grappe ». À l'intérieur d'une grappe, les blocs fonctionnels peuvent être branchés entre eux par l'entremise de topologies traditionnelles comme un bus. La Figure 2-2 montre un réseau sur puce ayant quatre nœuds. Sur cette figure, il est possible de remarquer qu'un des nœuds est une grappe de blocs fonctionnels interconnectés par l'entremise d'un bus local.

Les blocs fonctionnels ne sont jamais directement connectés sur le NoC. Ils sont plutôt connectés à un adaptateur réseau (NI) qui est, à leur tour, connecté sur le NoC. Comme le nom le suggère, les adaptateurs réseau sont ceux qui sont responsables d'adapter les communications. L'utilisation d'adaptateur permet la séparation des communications du traitement ainsi que la réutilisation des blocs fonctionnels.

À l'intérieur d'un NoC, on retrouve des commutateurs réseaux, du nom anglais *switch*, ainsi que des médiums de communication. Les commutateurs réseaux sont les points d'entrées du NoC et son directement branché avec un adaptateur réseau. Les commutateurs réseaux s sont tous branchés sur les médiums de communication formant ainsi la topologie du NoC. Le médium de communication est souvent appelé « canal de communication ». Il peut y avoir plusieurs canaux de communication dans le NoC. Ces canaux de communication sont généralement paramétrables selon les besoins de l'application. Par exemple, on peut déterminer la largeur du canal ainsi que le nombre de canaux désirés.

Les blocs fonctionnels échangent des messages afin d'effectuer la tâche pour laquelle le système est conçu. Le terme « paquet » est utilisé pour décrire ce qui circule à l'intérieur d'un NoC. Chaque paquet contient un entête ainsi que les données utiles. Les blocs fonctionnels envoient les données utiles ainsi que l'identifiant pour lequel le message est destiné. Le paquet est créé par l'adaptateur réseau. Ceci est une application directe du modèle OSI.

Le commutateur réseau est celui qui se charge d'aiguiller les paquets vers la bonne destination par l'entremise de l'entête. Le commutateur réseau doit décoder l'entête afin de détermine le nœud de destination. Afin de déterminer le chemin par lequel le paquet va emprunter, le commutateur réseau applique sa politique de routage.

Le concept de réseau sur puce existe puisqu'il permet de résoudre le problème des communications des topologies traditionnelles (bus et point à point) lorsqu'on doit interconnecter beaucoup de blocs fonctionnels.

En effet, dans un système orienté bus, le bus devient le goulot d'étranglement du système limitant ainsi le débit. Dans un système multi-bus, l'arbitration devient complexe et peut mener à des interblocages (de l'anglais *deadlocks*). Dans un système orienté point à point, le nombre de liens augmente de façon quadratique menant à un problème d'allocation.

Heureusement, l'introduction des réseaux sur puce résout ses problèmes [15] en proposant une nouvelle topologie de communication extensible.

2.1.1 Architecture de réseau sur puce

Tel que discuté préalablement, l'interconnexion des commutateurs réseaux forme la topologie du NoC. L'interconnexion peut être homogène ou hétérogène. Une topologie homogène signifie qu'une seule topologie de NoC est utilisée. Une topologie homogène est généralement uniforme et régulière. À l'inverse, une topologie hétérogène signifie qu'il y a présence d'au moins deux types différents de topologie. Une topologie hétérogène est plus ou moins régulière. L'utilité d'une topologie irrégulière est qu'il est possible de l'optimiser pour une application spécifique [31, 43].

Dans la littérature, les topologies de réseau couramment utilisées sont le maillage, l'anneau et l'arbre. Une topologie en anneau signifie que les éléments sont disposés en anneau tandis qu'une topologie en arbre dénote une arborescence (parent et fils). Le maillage est remarquable par son apparence de plan hippodamien (en quadrillé). La Figure 2-3 (de a à c) montre une topologie en arbre, en maillage et finalement, en anneau.

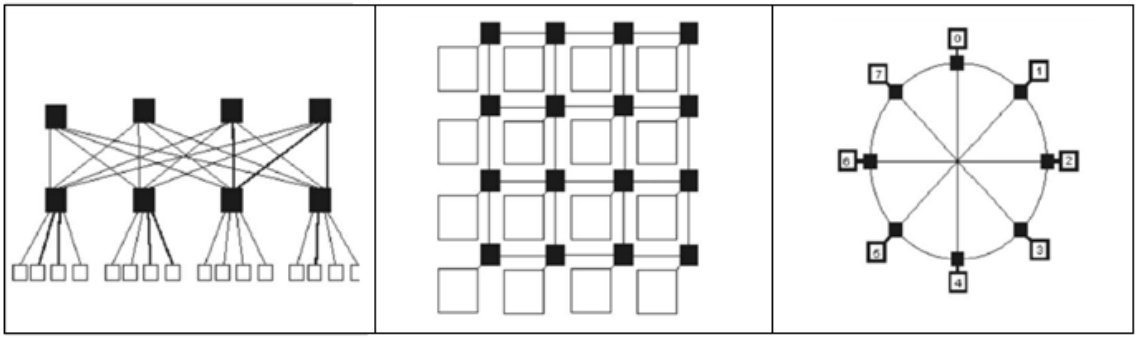

Figure 2-3 : Architectures de réseau sur puce [34]

À titre d'exemples d'architectures de réseaux sur puce bien connues, mentionnons : SPIN [32], CLICHÉ [21] et Octagon [33]. L'architecture SPIN s'apparente à une topologie en arbre tandis que l'architecture CLICHÉ s'apparente à une topologie en maillage (mesh) et finalement, l'architecture Octagon s'apparente à une topologie en anneau.

En observant uniquement la topologie (Figure 2-3), il est difficile de prédire laquelle est la plus adéquate pour une tâche particulière. Il faut donc comparer les performances de celles-ci. Dans cette optique, les auteurs de [34, 38] s'intéressent à l'évaluation de performance ainsi qu'aux compromis des différentes architectures de réseaux sur puce. Plus particulièrement, les auteurs de [34] présentent une méthodologie d'évaluation basée sur des paramètres considérés critiques par les experts du domaine. Ces paramètres sont le débit des messages, la latence, l'énergie dissipée et finalement l'espace requis. Les auteurs expliquent et définissent rigoureusement chaque paramètre. C'est ainsi qu'ils comparent six architectures de réseaux sur puce : SPIN [32], CLICHÉ [21], Torus [35], Folded torus [36], Octagon [33] et finalement BFT (Butterfly Fat-Tree) [37]. Les résultats ainsi présentés sont utiles pour les concepteurs dans leurs quêtes d'une architecture plus performante. Ils seront ainsi capables de connaître les limites et les compromis des réseaux sur puce.

2.1.1.1 CLICHÉ

Dans [21], les auteurs proposent la méthodologie BPS « Backbone-Platform-System » s'intéressant à la réutilisabilité des plate-formes. Afin de montrer leur méthodologie en action, les auteurs ont créé le réseau sur puce nommé CLICHÉ. La méthodologie BPS ne sera pas présentée ici. Nous nous intéressons davantage à l'architecture CLICHÉ.

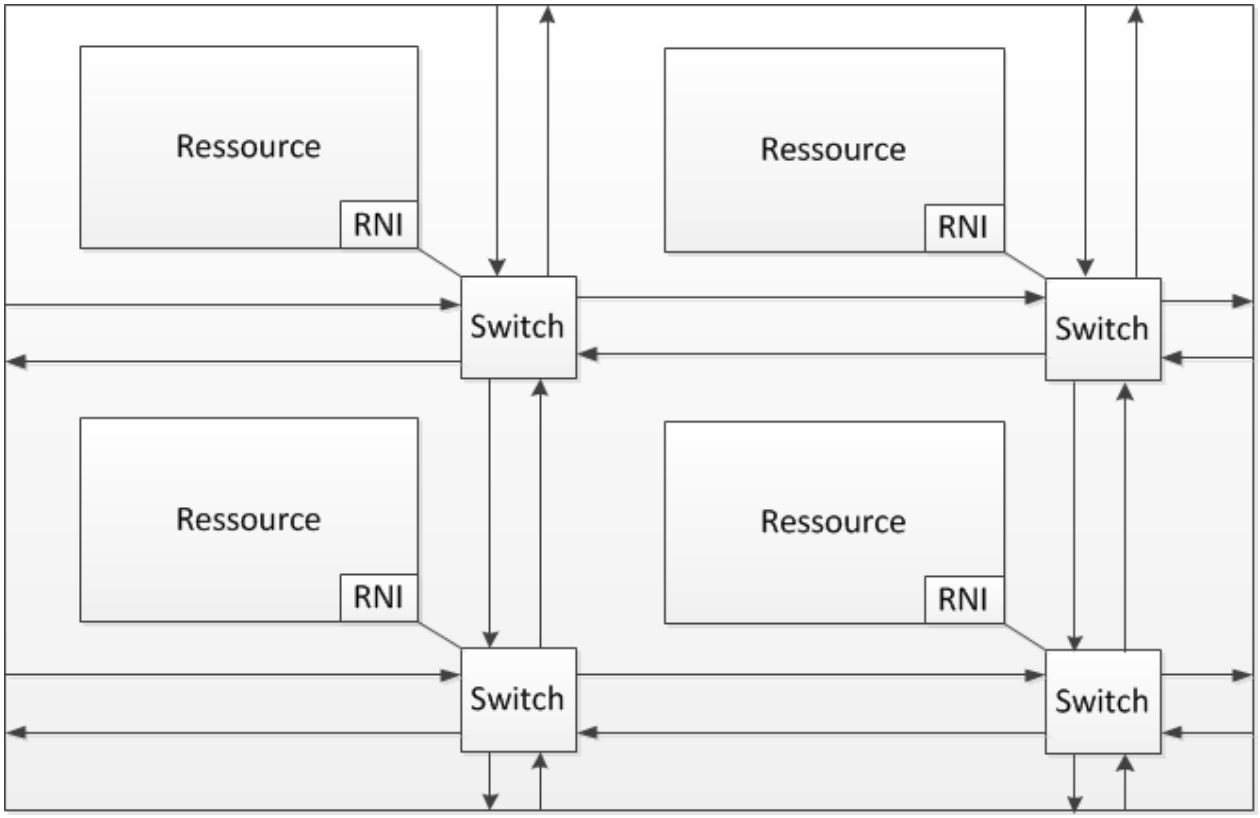

Figure 2-4 : CLICHÉ

CLICHÉ est un acronyme provenant de « Chip-Level Integration of Communication heterogeneous Elements » désignant une architecture de réseau sur puce reposant sur la topologie en maillage. Comme le nom le suggère, une telle architecture est conçue pour une utilisation hétérogène de blocs fonctionnels (multiprocesseurs, bloc matériel configurable, etc.). La Figure 2-4 présente l'architecture CLICHÉ ayant 16 ressources. Chaque ressource communique avec la *switch* par l'entremise d'un RNI « resource network interface ». Les *switch* sont branchés aux quatre voisins connexes. Les ressources peuvent fonctionnée de façon asynchrone par rapport aux autres ressources. La synchronisation des ressources a lieu lors d'envoi de messages.

Le RNI implémente les quatre couches du modèle OSI [19] (physique, liaison de données, réseau et transport) tandis que les *switch* implémentent les trois couches les plus basse (physique, liaison de données et réseau). Par conséquent, une ressource peut être utilisé dans CLICHÉ si

cette dernière intègre un RNI et qu'une place est libre dans le l'architecture, car le RNI effectue les transformations nécessaires afin de permettre le branchement dans CLICHÉ.

2.1.1.2 Rotator-On-Chip

Le Rotator-On-Chip, communément appelé RoC, est une architecture de réseau sur puce homogène basée sur la topologie en anneau. Le RoC a fait l'objet de diverses publications [3, 42] et émerge du brevet [40] détenu par STMicroelectronics. Le brevet [40] décrit une architecture de réseau sur puce appelée Hyper-ring-on-chip (HyRoC).

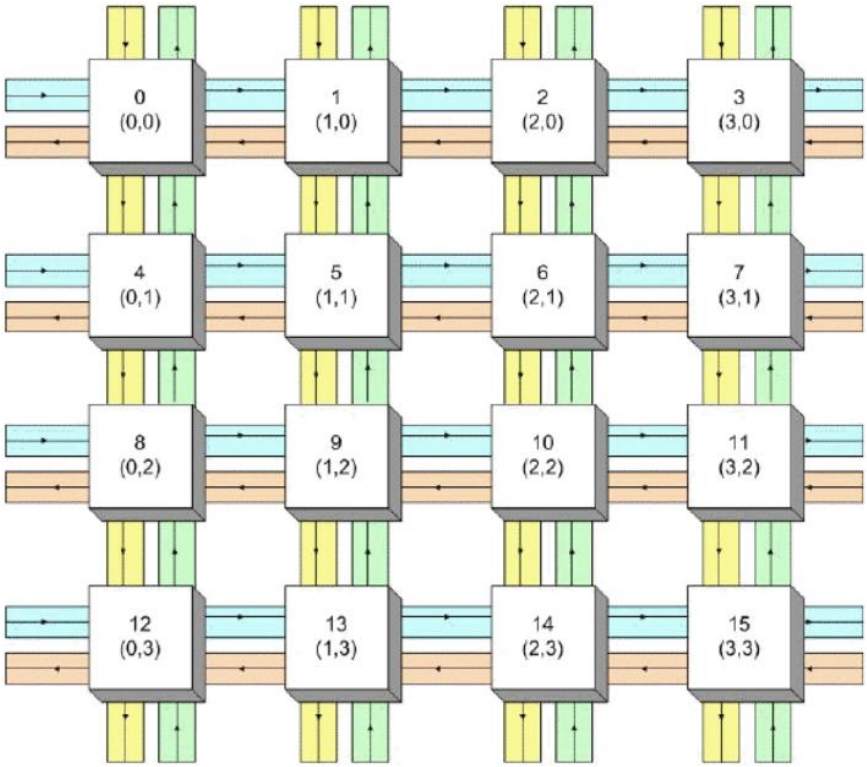

Figure 2-5 : Hyper-ring-on-chip

La Figure 2-5, tiré de [40], présente cette architecture sur laquelle on remarque que les ressources sont disposées en maillage 4x4. Les ressources, notées 0 à 15, sont branchées sur une interface réseaux (NI) laquelle est branché sur les anneaux connexes (lignes bleues et rouges, ainsi que les colonnes jaunes et vertes). Autrement dit, les ressources communiquent entre elles

par le biais d'anneaux circulaires. Il existe un anneau de communication pour chaque colonne et pour chaque rangée. Par conséquent, il existe 16 anneaux sur la Figure 2-5.

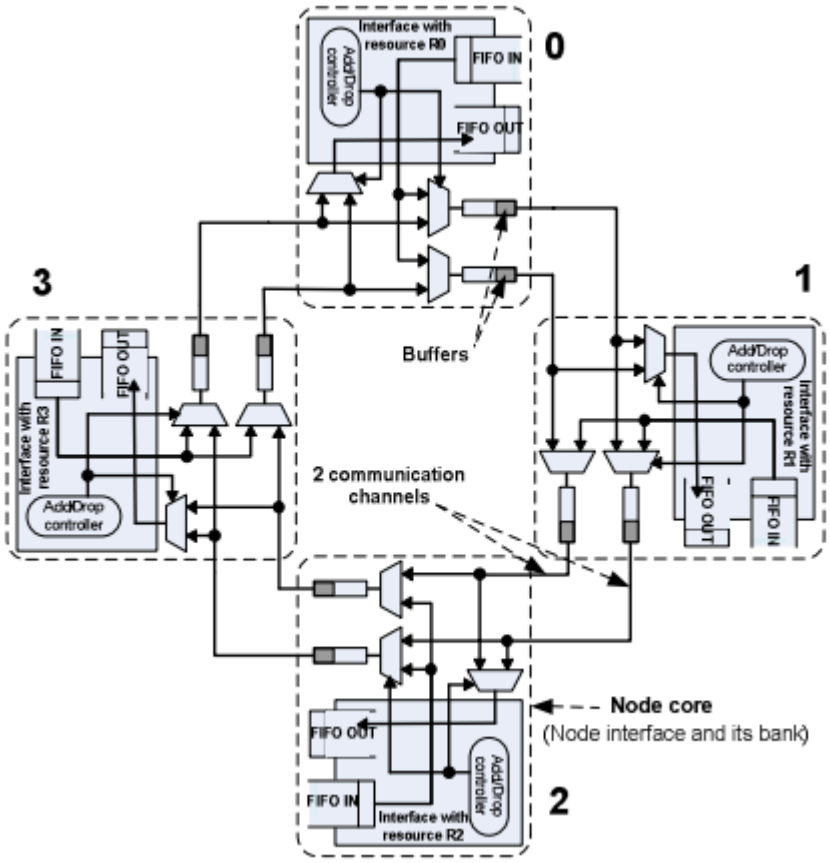

Figure 2-6 : Rotator-On-Chip

Le Rotator-On-Chip est un fragment du Hyper-ring-on-chip. Essentiellement, le RoC est un anneau parmi les nombreux anneaux constituant le HyRoC. La Figure 2-6, tirée de [3], montre le RoC interconnectant les interfaces réseaux (NI) à l'aide de deux anneaux circulaires unidirectionnels tournant dans le sens horaire. Les NI sont présentés comme l'élément principal d'un nœud (« Node core »), ces derniers étant numérotés de 0 à 3.

Dans [3, 42], les auteurs présentent l'implémentation RTL du Rotator-On-Chip. Un NI est constitué d'une fifo (de l'anglais *first in, first out*) d'entrée, une fifo de sortie et d'un contrôleur

d'injection et d'éjection de paquet. Par la suite, les auteurs expliquent la procédure par laquelle les paquets sont injectés et éjectés ainsi que la stratégie mise en place pour garder les informations sur la disponibilité des canaux (via des signaux parallèles au canal de communication). Le contrôleur d'injection et d'éjection de paquets [3] garantie que les paquets ainsi reçus par les destinateurs seront dans l'ordre. De plus, ce même contrôleur peut injecter et éjecter un paquet à chaque coup d'horloge si la fifo d'entrée à la source contient des données et que la fifo de sortie à destination a des places de libres. Pour assurer qu'il y a toujours des places libres à destination, un mécanisme de « contrepression », du nom anglais « backpressure », permet au contrôleur d'un nœud d'indiquer aux autres contrôleurs de nœud s'il est en contrepression. Un contrôleur devient en contrepression lorsque sa fifo de sortie est pleine ou lorsqu'elle dépasse un seuil critique. Lorsqu'un contrôleur est en contrepression, il envoie un signal en diffusion générale aux autres contrôleurs afin qu'ils cessent leurs émissions vers ce dernier. Le signal émis ne passe pas par les canaux de communication. Il s'agit plutôt d'un registre partagé contenant les informations de *backpressure* des NI. Le contrôleur demeure en contrepression tant et aussi longtemps que la fifo dépasse le seuil critique.

Il existe trois extensions de l'implémentation du Rotator-On-Chip. La première est le support pour un RoC bidirectionnel. Un RoC bidirectionnel signifie qu'il y a un anneau dans le sens horaire et antihoraire. Une telle implémentation permet d'obtenir un meilleur débit ainsi que de diminuer la latence. La deuxième extension est le support pour la qualité de service. La qualité de service est utile lorsqu'on désire isoler du trafic prioritaire du trafic moins prioritaire (par exemple le son et l'image dans un traitement vidéo). Pour chaque qualité de service, une fifo d'entrée est ajoutée. La dernière extension du RoC est le support pour diminuer/augmenter le chemin des données afin de diminuer/augmenter l'espace requis. Dans l'implémentation de base, le chemin des données est fixé à 96 bits.

Par la suite, les auteurs se sont intéressés à la caractérisation ainsi que l'espace consumé sur une puce FPGA. Lors de la caractérisation, les auteurs remarquent que l'ajout de canaux permet une augmentation linéaire du débit, mais qu'à un certain point, le débit sature. Un autre résultat intéressant est la comparaison de l'espace consumé de l'architecture. En effet, les auteurs

ont synthétisé le RoC et l'ont comparé à un maillage 2D 4x4. La conclusion d'une telle comparaison est que le RoC est jusqu'à cinq fois plus petit qu'un maillage 2D 4x4.

2.1.1.3 Arteris

Arteris [41] offre des solutions de réseaux sur puce par le biais de leurs produits : FlexNoC, FlexWay et C2C (Chip to Chip Link). En plus des avantages des NoC, les solutions qu'offre cette compagnie ont comme but de supporter divers protocoles (AMBA, OCP, NIF et PIF), optimiser les latences pour une architecture spécifique et d'offrir une topologie flexible. En utilisant leur méthodologie, les concepteurs sont en mesure de sauver du temps, moins de fils, moins de congestion, plus facile de rencontrer la fréquence souhaitée (*timing closure*), d'économie de l'espace sur le die, moins de risque de problème d'ordonnancement.

2.1.2 Méthodologie de conception

Dans [14], les auteurs proposent une description unique pour les architectures de réseaux sur puce. Cette proposition est formée par trois paradigmes distincts. Le premier paradigme est l'infrastructure de communication qui tient compte de l'application cible, les commutateurs réseaux et des canaux. Le deuxième paradigme est celui des communications qui tient compte de l'algorithme de routage, l'utilisation des commutateurs réseaux voisins et de la commutation des paquets. Le dernier paradigme est celui de l'assignation (*mapping)* des blocs fonctionnels aux commutateurs réseaux. Puisque chaque paradigme définit une dimension, l'exploration architecturale des réseaux sur puce se fait dans un espace 3D. Il faut donc explorer chacune des dimensions de façon unitaire si l'on veut obtenir un NoC optimal pour une application donnée.

Dans [44], les auteurs s'intéressent au *application-specific network-on-chip* (ASNoC) afin de résoudre les limitations des topologies régulières de réseau sur puce. Dans un ASNoC, il n'y a pas de topologie fixe (topologie hétérogène). La topologie est plutôt définie par les besoins en communication, *floorplan* et le design des *switch*. Un ASNoC tente de maximiser les communications locales par l'entremise d'une grappe ayant un bus local. La Figure 2-7 montre un exemple d'un ASNoC. Les auteurs présentent une méthodologie automatique afin de générer une architecture « optimisée » ASNoC pour une application spécifique.

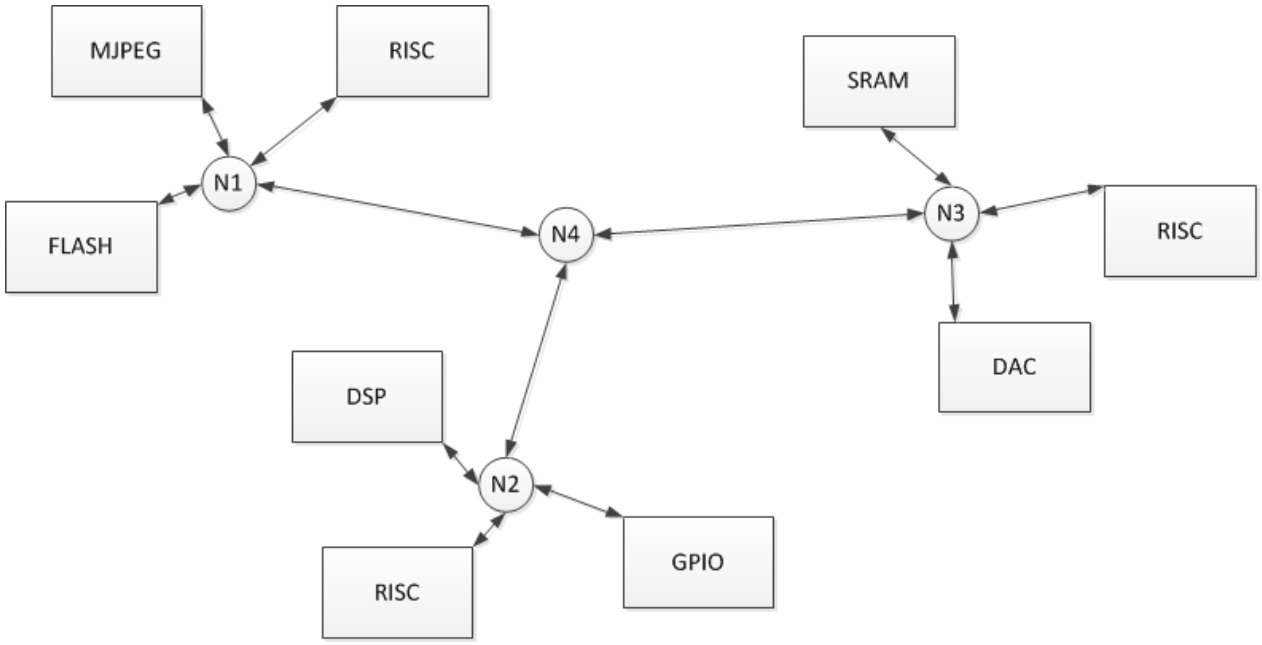

Figure 2-7 : ASNoC

La méthodologie présentée dans [44] comprend cinq étapes. La première étape consiste à une analyse des communications. L'analyse détermine les types de communication, la fréquence ainsi que le volume de données. La deuxième étape génère une architecture ASNoC selon une librairie de composante précise au cycle près (Cycle accurate). La troisième étape consiste à une estimation du *floorplan* afin de réduire la longueur des interconnections. La quatrième étape effectue une analyse de performance et compare ces résultats avec les requis et spécifications d'entrées. Si les requis et spécifications ne sont pas rencontrés, le processus doit recommencer depuis le début. La dernière étape consiste à une estimation de la puissance et de l'espace requise.

Finalement, les auteurs appliquent leur méthodologie à l'application H.264. Il en résulte que l'ASNoC généré utilise 39% moins d'énergie, 59% moins d'espace et permet une accélération d'un facteur deux comparé à une architecture homogène 2D en maillage.

2.2 Niveaux d'abstractions

L'acronyme ESL signifie « Electronic System Level » et représente une méthodologie de conception de système qui met l'emphase sur l'utilisation de modèle abstrait (modèle haut niveau) [9]. Il existe plusieurs niveaux d'abstraction que nous allons couvrir dans un moment, mais avant tout, nous allons positionner la librairie SystemC.

En soi, SystemC [11] n'est pas un langage de programmation, mais plutôt une librairie de simulation écrit en C/C++. Il est important de mentionner que SystemC est l'application concrète de la méthodologie ESL, car elle permet l'abstraction des modèles. L'utilité d'une telle librairie est qu'elle permet en autres de modéliser des blocs fonctionnels à différents niveaux d'abstraction et de simuler le système. Puisqu'il s'agit d'une simulation, il est possible d'abstraire certains détails d'implémentation augmentant ainsi l'exécution de la simulation dans le but de faire une exploration architecturale [28].

Voyons maintenant en détail les divers niveaux d'abstraction. Dans [28], les auteurs identifient deux types d'abstraction possible : communication et calcule. Chaque abstraction a trois degrés de liberté : *un-timed*, *approximate-time* et *cycle-timed*. Le nouveau standard TLM2 propose le model de programmation *loosely-timed* [47] se situant à mi-chemin entre *un-timed* et *approximate-timed*. L'utilité du *loosely-timed* est qu'il permet d'accélérer davantage la simulation principalement à cause du découplage temporel.

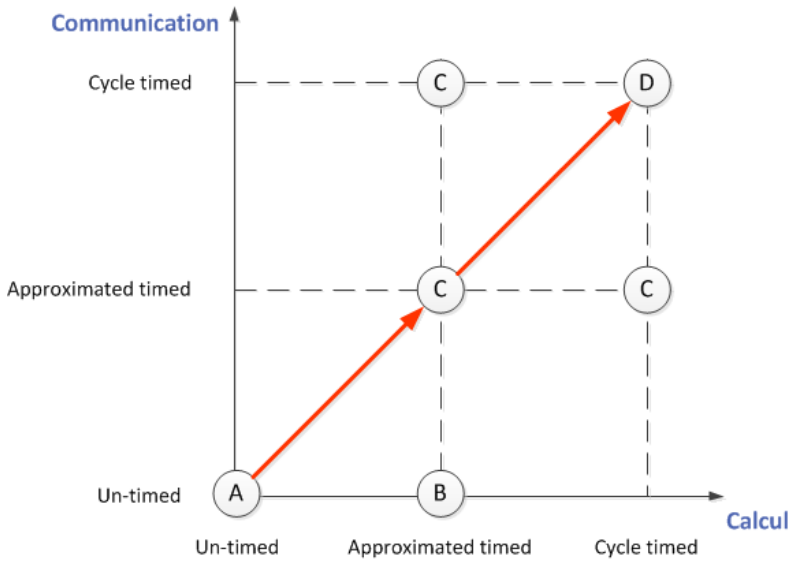

Figure 2-8 : Modèles abstraits

La Figure 2-8, tiré de [28], montre les différents niveaux d'abstraction ainsi que le chemin de raffinement. L'axe l'horizontale présente l'abstraction des calculs tandis que l'axe vertical présente le niveau d'abstraction des communications. Le raffinement a lieu lorsqu'on se déplace le long des axes. Sur la Figure 2-8, la flèche rouge représente le chemin couramment utilisé lors du raffinement.

Lorsqu'on se déplace de gauche vers la droite, on raffine les calculs en y ajout de la précision. Puisqu'il s'agit de l'axe horizontal, on raffine l'algorithme de calcul du modèle. Cette précision ainsi rajoutée est l'ajout de latence remarquable par le raffinement de l'algorithme effectué.

Lorsqu'on se déplace du bas vers le haut, on raffine les communications en y ajout de la précision. Puisqu'il s'agit de l'axe vertical, on raffine les communications du modèle. Cette précision ainsi rajoutée est remarquable par le raffinement des interfaces de communications. Par exemple, pour SystemC, ceci se traduit le remplacement des canaux abstraits (canaux hiérarchiques) par un canal de communication complet.

Toujours à l'aide de la Figure 2-8, le point A correspond au niveau d'abstraction UTF (Untimed functional) et est utilisé lors de la validation fonctionnelle d'un système. Dans ce niveau d'abstraction, on utilise les canaux abstraits (canaux hiérarchiques).

Le point B correspond au niveau d'abstraction TF (Timed Functional) où le calcul effectué par le modèle est raffiné en y ajoutant de la latence.

Le point C correspond au niveau d'abstraction BCA (Bus Cycle Accurate) où la communication effectuée par le modèle est raffinée. Ceci est traduit par l'ajout des latences dans le protocole de communication ou par le remplacement des canaux abstraits par un canal de communication complet. Dans [30], les auteurs divisent le niveau d'abstraction BCA en deux : PA-BCA (Pin-Accurate Bus Cycle Accurate) et T-BCA (Transaction based Bys Cycle Accurate). La différence réside au niveau du canal de communication : PA-BCA utilise des signaux élémentaires tandis que T-BCA utilise des interfaces abstraites.

Finalement, le point D correspond au niveau d'abstraction RTL (Register-transfer level). Ceci est le dernier niveau d'abstraction possible. Un modèle se situant dans le niveau d'abstraction RTL ne possède aucune interface abstraite, mais plutôt des canaux primitifs SystemC (canal de communication complet). Les algorithmes de calculs sous forme chemin de données et machine à états sont précis intégrant les latences requises. Un modèle se situant au niveau d'abstraction RTL est en fait une représentation un pour un du même modèle écrit dans un langage de description matériel.

2.3 Adaptateur

Un adaptateur est un dispositif permettant à deux équipements différents de fonctionner ensemble. Dans les systèmes embarqués, les adaptateurs sont utilisés afin d'adapter les protocoles de communication. Dans [29], les auteurs présentent une méthodologie appelée « Interface-Based Design ». Ils définissent un *token* comme étant une communication complète entre deux ou

plusieurs blocs fonctionnels. Le détail de leur méthodologie est divisé en deux points et est applicable à chaque *token* :

1. Séparation du comportement des communications
2. Raffinement hiérarchique progressif

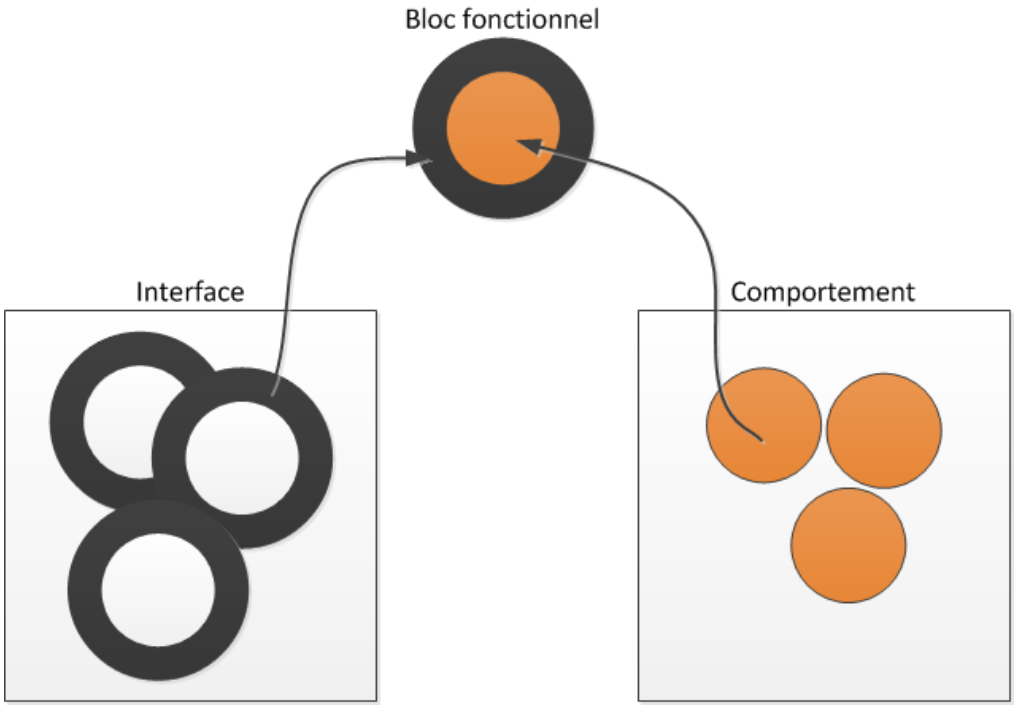

Figure 2-9 : Architecture à base d'interface

En plus d'adapter les protocoles de communication, les adaptateurs permettent de séparer le comportement (traitement) des communications. La Figure 2-9, tiré de [29], résume bien l'idée des auteurs. Sur cette figure, on voit que le comportement (traitement) est encapsulé par les interfaces. L'union entre le comportement et l'interface forme le bloc fonctionnel lequel est utilisé dans un système complet.

Le raffinement hiérarchique progressif est en fait le raffinement des interfaces. À haut niveau, l'interface est en fait l'échange de simple pointeur. Lorsqu'on raffine les modèles, l'interface abstraire est littéralement remplacer par une interface ayant un maître et un esclave. La

dernière partie du raffinement hiérarchique progressif se produit lorsque les blocs sont synthétisés. Lors de la synthèse, la partie maître de l'interface est synthétisée avec le bloc émetteur et la partie esclave de l'interface est synthétisée avec le bloc récepteur.

2.4 Outil haut niveau

Un outil haut niveau est un outil utilisant la méthodologie ESL « Electronic System Level ». Par conséquent, les outils haut niveau sont des outils de simulation utilisant une librairie de simulation. La librairie de simulation SystemC est probablement la plus connue. Cette libraire est maintenant un standard IEEE (IEEE 1666).

2.4.1 SPACE

La plate-forme virtuelle SPACE, tel que présenté dans [24, 25, 27], est un outil de simulation haut niveau écrit en C/C++ utilisant la librairie de simulation SystemC permettant la cosimulation (la simulation de composante matérielle et logicielle). L'outil SPACE vient avec un vue graphique en plus d'un engin d'analyse, appelé *SpaceMonitor*. Dans [24, 25, 27], les auteurs présentent leur méthodologie ainsi que leur stratégie de simulation de logiciel embarqué. Leur méthodologie est divisée en deux points.

La première étape consiste à décrire le système en bloc fonctionnel, appelé module, et de valider le fonctionnement du système dans une plate-forme UTF. Lors de la validation du système, les modules sont branchés à un bus UTF lequel peut être vu comme un *crossbar*.

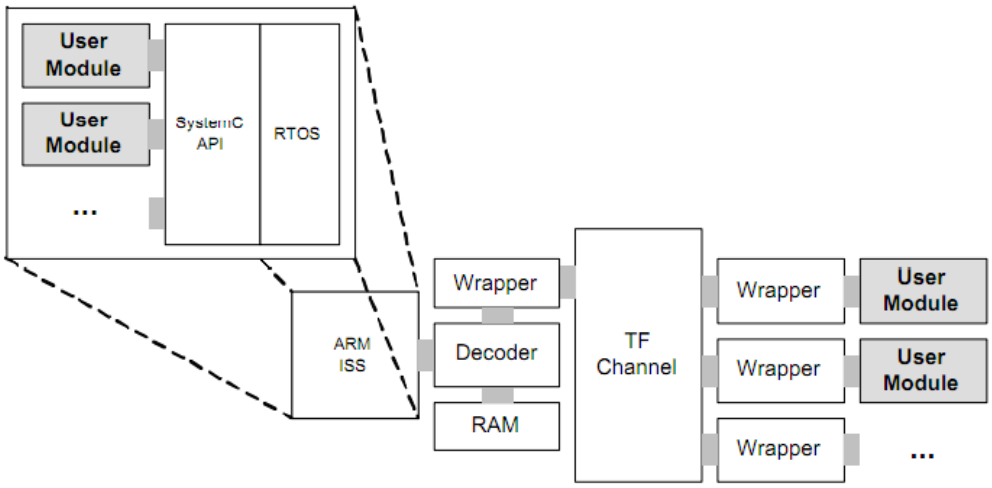

Figure 2-10 : SPACE

La deuxième étape est de raffiner le modèle vers une représentation TF et BCA. La Figure 2-10 montre une architecture TF ayant un ISS. Durant cette étape, les modules peuvent être placés en logiciel ou en matérielle. Puisque SystemC ne supporte pas l'exécution de logiciel embarqué [25], les auteurs proposent l'utilisation d'un ISS développé en SystemC sur lequel est embarqué le module, un API traduisant les appels SystemC en appel OS, d'un RTOS et finalement d'un HAL.

2.4.2 SoCLib

La plate-forme virtuelle SoCLib [26] est un outil de simulation haut niveau pour les MPSoC. Cet outil repose sur la librairie de simulation SystemC et propose deux modèles d'abstraction pour chaque bloc fonctionnel. Ces modèles d'abstraction sont CABA (Cycle accurate/Bit accurate) et TLM-DT (Transaction Level Modeling with Distributed Time). Le principal avantage de la libraire SoCLib est qu'il permet l'interopérabilité entre tous les blocs fonctionnels, car tous les blocs supportent le protocole VCI (Virtual Component Interface). En plus d'offrir un large éventail de blocs fonctionnels (7 ISS, 3 NoC, 3 Bus, etc.), SoCLib vient avec plusieurs outils complémentaires à la librairie. On y retrouve notamment l'outil DSX (Design Space Exploration) servant à l'exploration architecturale. Un autre outil est SystemCASS servant à simuler une architecture matérielle CABA jusqu'à 15 fois plus rapide que celui proposé par SystemC.

CHAPITRE 3 MÉTHODOLOGIE DE CONCEPTION POUR RÉSEAU SUR PUCE À BASE DE ROC

Dans ce chapitre, nous décrivons la réalisation ainsi que les choix qui ont mené à cette dernière. Cette réalisation consiste à intégrer un réseau sur puce dans l'outil SPACE tout en conservant les possibilités d'offres cet outil (co-design, raffinement sur puce, déplacement des blocs, etc.). Dans un premier temps, nous allons introduire le modèle de réseau sur puce ainsi que l'outil haut niveau retenu pour notre expérience. Par la suite, nous allons discuter des métriques d'évaluations nécessaires pour l'évaluation du modèle développé. Finalement, nous allons expliquer en détail le prototype développé ainsi que les requis pour celui-ci.

3.1 Réseau sur puce

Tel que discuté au chapitre 2, il existe de nombreux modèles de réseau sur puce dans la littérature. Nos critères de sélection sont la facilité de développement du modèle ainsi que les ressources disponibles (personne ressource, documentation, articles, etc.). Suite à la revue des travaux antérieurs portant sur les réseaux sur puce, les modèles retenus pour l'évaluation furent le Rotator-On-Chip [3, 42] et CLICHÉ [21]. Après une exploration approfondie des modèles, notre choix s'est arrêté sur le Rotator-On-Chip puisque : 1) il offre de bons compromis entre la performance, l'utilisation des ressources et l'Adaptabilité (i.e. au niveau évolutif) et 2) ce modèle a fait l'objet de plusieurs travaux à l'École Polytechnique et il offre ainsi une proximité de personnes ressources.

Lors de la section 2.1.1.2, nous avons vu que le RoC peut être étendu afin de supporter trois extensions (bidirectionnel, qualité de service, et diminution/augmentation du chemin des données). Nous avons décidé de modéliser le Rotator-On-Chip ayant les caractéristiques de base ainsi que toutes les extensions outre l'augmentation/diminution du chemin des données, tel que présenté dans [22]. Par conséquent, les paramètres de configuration pour le RoC sont les suivantes :

- Profondeur des fifos d'entrée
- Profondeur de la fifo de sortie

- Nombre de canaux (dans les deux directions)
- Nombre de nœuds
- Nombre de priorité (pour les fifos d'entrée uniquement)
- Valeur du backpressure

3.2 Choix d'une plate-forme virtuelle

Nous utilisons l'outil SPACE [24, 25, 27] dans le cadre de ce travail, car cet outil offre des avantages que nous désirons utiliser comme levier à la suite de notre intégration [39] (co-design, déplacement des blocs, analyse non-intrusive (matériel et logiciel), raffinement sur puce).

L'outil SPACE utilise l'approche en Y [16] permettant de lier une application à une architecture dans le but de rencontré les requis et spécifications initiaux. Si les requis ne sont pas rencontré, on réitère dans le processus. Nous allons utiliser cette approche pour notre architecture cible.

Finalement, puisque cet outil développé à l'École Polytechnique de Montréal, il sera possible d'avoir accès au code propriétaire facilitant ainsi la compréhension globale de l'outil, le développement ainsi que l'intégration de notre travail dans ce dernier.

3.3 Métriques d'évaluation pour la communication

Les métriques d'évaluations sont principalement utilisées afin d'obtenir des informations pertinentes sur la simulation afin de guider les itérations lors du cycle de développement de l'architecture selon les requis et spécifications initiaux. Nous utilisons également les métriques d'évaluation afin de valider les compromis de vitesse de simulation versus précision du niveau d'abstraction ciblé.

Il est difficile de généraliser les métriques d'évaluation à l'ensemble des réseaux sur puce, car chaque implémentation de modèle peut supporter un sous-ensemble de métrique d'évaluation.

Parfois, ces sous-ensembles peuvent être disjoints. Par exemple, dans le cas du Rotator-On-Chip, le modèle a comme caractéristique d'avoir deux sens de parcours (horaire et antihoraire). Ce concept n'est pas le cas pour un maillage. Par conséquent, il ne fait pas sens de généraliser le concept de sens horaire et antihoraire à l'ensemble des réseaux sur puce.

Mais afin de simplifier l'implémentation des métriques d'évaluation, nous avons décidé de cibler un modèle bien précis de réseau sur puce afin de développer les métriques d'évaluation. Par conséquent, les métriques proposées ci-dessous sont spécifiques au Rotator-On-Chip mais peuvent être également applicables à d'autres modèles lorsque les modèles présentent des caractéristiques similaires.

Nous nous sommes inspirés de [34] afin de déterminer les métriques de performances utiles. Nous avons déterminé que les métriques suivantes sont nécessaires :

- Utilisation des fifos
- Latence des paquets
- Nombre de paquets émis, reçus et perdus

Afin d'implémenter ses métriques, deux tâches doivent être effectuées. Premièrement, il faut étendre l'infrastructure d'analyse de performance proposée par l'outil haut niveau afin d'ajouter le support des métriques proposées. Cette analyse de performance se fait de façon non-intrusif, car l'analyse est effectuée parallèlement à la simulation. Deuxièmement, il faut ajouter les instructions de surveillance à l'intérieur du modèle haut niveau lors de son développement.

3.3.1 Utilisation des fifos

Les fifos sont une partie essentielle au réseau sur puce. La plupart des architectures de réseau sur puce les utilisent. Il est important de faire l'analyse des fifos, car une fifo pleine veut généralement dire que le modèle est en saturation. Inversement, une fifo presque vide veut dire que le modèle suffit à la demande.

$$nb_{fifo} = 2 * (nb_{prio} + 1)$$

Le nombre de fifo que l'on retrouve par nœud est donné par l'équation ci-dessus. Par conséquent, on trouve au minimum quatre fifos par nœud dont deux dans le sens horaire et deux dans le sens antihoraire. Pour chaque fifo, les informations suivantes sont disponibles :

- Utilisation moyenne des fifos
- Utilisation maximale des fifos

$$F_{avg} = \frac{\sum_{i=1}^{E} F_i}{E}$$

$$F_{max} = \max(F_1, F_2, \dots, F_E)$$

où F_i est le nombre d'éléments dans la fifo à un moment donné, et

E est le nombre d'échantillonnages

Chaque ajout et retrait d'élément dans la fifo doit être relevé par le modèle. Les informations suivantes sont nécessaires lors de l'échantillonnage:

- Le nœud concerné
- S'agit-il de la fifo d'entrée ou de sortie?
- S'il s'agit de la fifo d'entrée, quelle est la fifo de priorité utilisée?
- Combien d'éléments sont présents dans la fifo actuelle
- Direction concernée (horaire ou antihoraire)

Lorsque la simulation se termine, l'engin d'analyse effectue la moyenne et trouve la valeur maximale d'élément contenu dans la fifo et ce, pour chaque fifo. Les informations recueillies sont sauvées en vue de fournir un rapport à la toute fin de la simulation.

3.3.2 Latence des paquets

La latence des paquets est la métrique de performance la plus utile, car elle démontre facilement la contention dans le modèle. Il est important d'effectuer l'analyse de la latence des paquets, car elle permet de réorganiser les nœuds afin de minimiser la latence (meilleure proximité). La latence est affichée en termes de cycle de simulation. Les informations suivantes sont mémorisées pour chaque paire de communications :

- Latence minimale
- Latence maximale
- Latence moyenne

La latence minimale est simplement la plus petite latence obtenue par un paquet partant du nœud source et allant au nœud de destination. Inversement, la latence maximale est la plus grande latence obtenue par un paquet partant du nœud source et allant au nœud de destination.

$$L_{min} = \min(L_1, L_2, \dots, L_P)$$

$$L_{max} = \max(L_1, L_2, \dots, L_P)$$

où P est le nombre de paquets reçus par nœud.

La latence moyenne est probablement la métrique la plus intéressante, car elle dessine le portrait général de la saturation dans le modèle. En effet, si la valeur de la latence moyenne s'approche de la valeur de la latence maximale, on peut conclure que le système est en saturation. Inversement, si la valeur de la latence moyenne tend vers celle de la latence minimale, on peut conclure que le modèle répond aux besoins du système. Tel que décrit dans [34], la latence moyenne peut être vue par l'équation suivante :

$$L_{avg} = \frac{\sum_{i=1}^{p} L_i}{P}$$

où P est le nombre de paquets reçus par nœud, et

$L_i$est la latence d'un paquet

Pour produire les informations sur la latence, chaque ajout et retrait de paquet doit être relevé par le modèle. Les informations suivantes sont nécessaires lors de l'échantillonnage:

- Source (le module qui initie la communication)
- Destination (le module pour lequel le paquet doit être transmit)
- Priorité du paquet
- Temps de la simulation

À l'instar de la métrique de l'utilisation des fifos, l'engin d'analyse effectue les traitements nécessaires pour la production du rapport lorsque la simulation se termine. Les traitements requis sont de déterminer la moyenne, le minimum ainsi que le maximum parmi les échantillons.

3.3.3 Nombre de paquets émis, reçus et refusés

Pour chaque nœud, l'engin d'analyse détermine le nombre de paquets émis, reçus et refusés. Le nombre de paquets refusés est le nombre de paquets désirant entrer dans le modèle, mais sans succès. L'unique cause d'un refus est que la fifo d'entrée est pleine. Lorsqu'un paquet ne peut entrer dans le modèle, c'est à l'application d'effectuer un nouvel essai.

Les échantillons obtenus par la métrique de la latence des paquets permettent, par ricochet, de déterminer le nombre de paquets émis et reçus pour chaque nœud. Afin d'obtenir le nombre de paquets refusé dans le modèle, nous avons ajouté une instruction d'analyse dans le modèle. Cet échantillon requiert uniquement le nœud pour lequel le paquet est refusé.

Lorsque la simulation se termine, l'engin d'analyse effectue la sommation des échantillons pertinents et inscrit les valeurs dans le rapport.

3.4 Composants de l'architecture, assignation et raffinement

Dans cette sous-section, nous allons présenter les composants de l'architecture requis, l'assignation d'une application sur une architecture cible et finalement, son raffinement.

Tout d'abord, la librairie SPACE est formée de plusieurs composantes. Les composants nécessaires sont les modèles de réseaux sur puce (haut et bas niveau) ainsi que les adaptateurs requis (haut et bas niveau). Par la suite, nous allons présenter une assignation de l'application cible sur les composants existants de la librairie SPACE et ceux dûment conçu. Finalement, nous allons raffiner l'architecture cible vers une puce FPGA.

3.4.1 Développement des modèles

Le but de la phase de développement des modèles consiste à créer un modèle haut niveau et bas niveau du Rotator-On-Chip. Le modèle haut niveau sera utilisé lors de la simulation dans l'outil SPACE tandis que le modèle bas niveau sera utilisé lors de l'implémentation vers une cible finale. En d'autres mots, le modèle bas niveau est l'implémentation réelle du RoC tandis que le modèle haut niveau est son abstraction dans un environnement de simulation.

3.4.1.1 Haut niveau

La première étape lors du développement d'un modèle haut niveau consiste à faire le choix du niveau d'abstraction désiré. Les critères de sélection du niveau d'abstraction furent la précision du modèle, le recours à des interfaces haut niveau et l'abstraction des types de données. Le niveau d'abstraction BCA, se situant à mi-chemin entre un niveau UTF et RTL, a été retenu comme niveau d'abstraction. Plus précisément, nous utiliserons le niveau d'abstraction T-BCA, car nous désirons utiliser des interfaces hiérarchiques.

Il est possible de rendre notre modèle T-BCA précis en ajoutant les délais des communications lors de conception de ce dernier. Dans [3], nous avons observé que l'injection de

paquet dans le RoC prend deux cycles lorsque les fifos d'entrées ne sont pas pleines. Une fois le paquet à l'intérieur du RoC, chaque déplacement entre les nœuds prend un cycle. Finalement, l'éjection du paquet hors du RoC prend deux cycles lorsque la fifo de sortie n'est pas pleine. Conséquemment, le temps minimal pour une communication entre deux nœuds dont la distance est de 3 nœuds est de 7 cycles.

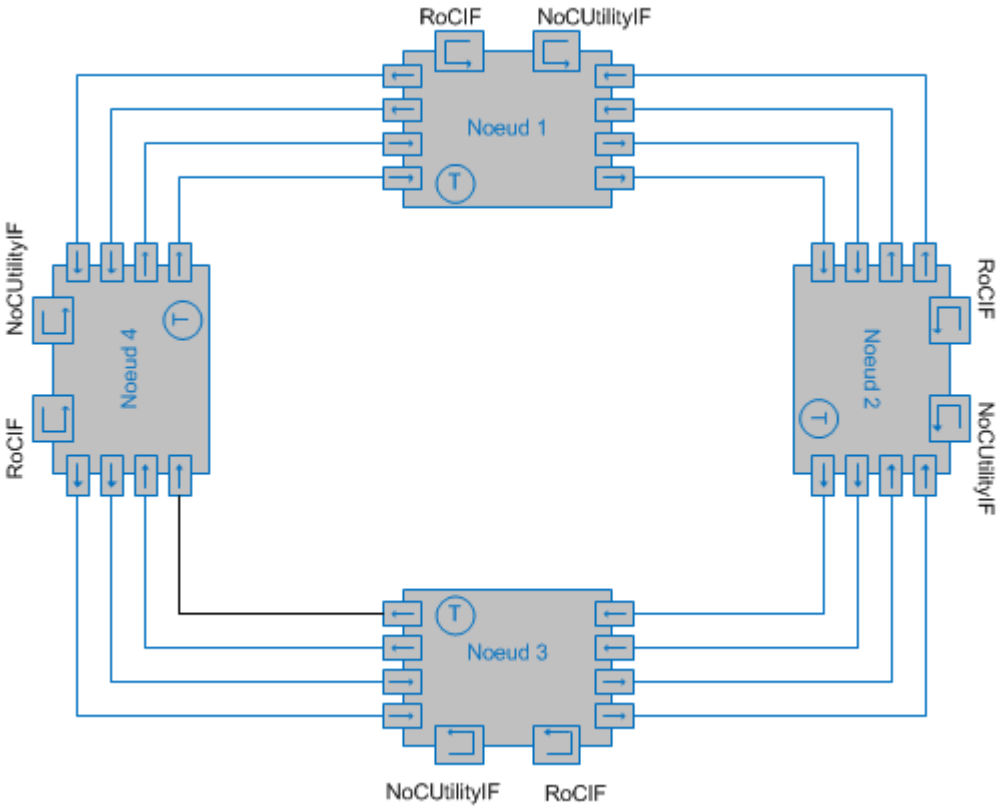

Figure 3-1: Exemple du Rotator-On-Chip modélisé à haut niveau

Nous avons utilisé la librairie SystemC [11] afin de réaliser le modèle haut niveau. La Figure 3-1 montre le portrait général du modèle haut niveau ayant quatre nœuds et deux canaux dans chaque direction. Chaque nœud possède deux interfaces haut niveau, un *thread* SystemC effectuant la logique d'injection et d'éjection de paquets. Les paramètres de configuration du modèle haut niveau se font lors de l'instanciation du modèle par le biais des arguments.

```
RoC(    sc_module_name Name, unsigned char ucBusID, unsigned int uiFifoInSize,
        unsigned int uiFifoOutSize, unsigned int uiNbChannels, unsigned int uiNbNodes,
        unsigned int uiNbPriorities, unsigned int uiBackpressure, const char* zMappingList);
```

Figure 3-2: Constructeur du Rotator-On-Chip à haut niveau

Le Figure 3-2 présente les arguments requis pour l'instanciation d'un Rotator-On-Chip dans un environnement haut niveau. Voici, dans l'ordre, une explication de chacun des arguments :

1. Nom de l'instance
2. Identifiant unique
3. Profondeur des fifos d'entrée
4. Profondeur de la fifo de sortie
5. Nombre de canaux (dans les deux directions)
6. Nombre de nœuds
7. Nombre de priorité (pour les fifos d'entrée uniquement)
8. Valeur du *backpressure* (dois être égale ou inférieure à la valeur de la profondeur de fifo de sortie)
9. Chemin vers le fichier texte indiquant l'assignation des modules sur les nœuds

```
0:12;
1:13;
2:14;15
3:16;
4:17;
```

Figure 3-3 : Exemple d'assignation pour le Rotator-On-Chip à haut niveau

Le Figure 3-3 montre un exemple d'assignation (*mapping*) des modules sur le NoC. Chaque ligne dans le fichier représente l'assignation pour un nœud donné. Le nœud est le chiffre avant le symbole « : » tandis que les chiffres situés après ce symbole et délimités par le

symbole « ; » indiquent les identifiants de module associé à ce nœud. En se fiant à la Figure 3-3, sur le nœud 2, on y retrouve les modules 14 et 15.

Toujours à l'aide de la Figure 3-1, chaque nœud possède deux interfaces haut niveaux. La première interface se nomme *RoCIF* et est utilisée pour l'injection et l'éjection de paquet de type *RoCTransaction* via l'appel aux méthodes *WriteRoC* et *ReadRoC*. Cette interface est spécifique pour le RoC et ne peut être générique à d'autres réseaux sur puce. Dans chacune des méthodes de l'interface, on trouve un appel à *wait(1),* provenant de la librairie de simulation SystemC, indiquant la latence de l'opération. Dans le cadre de l'ajout d'un nouveau réseau sur puce, cette nouvelle architecture de réseau sur puce devra définir sa propre interface ainsi que son propre type de paquet.

La deuxième interface, nommée *NoCUtilityIF*, est présente sur tous les modèles haut niveaux de réseau sur puce et possède une seule méthode appelée *FindNodeId* servant à indiquer sur quel nœud se trouve un module donné. Les adaptateurs se branchent sur cette interface et appel la méthode *FindNodeId* lorsqu'ils doivent construire le paquet à émettre. Les informations contenues dans le fichier d'assignation sont utilisées pour répondre aux appels de *FindNodeId*. La classe service appelée *NoCService*, contenue dans le NoC, est responsable de traiter les appels des méthodes provenant des interfaces *NoCUtilityIF*.

Les canaux, émulés par l'entremise de signaux SystemC *sc_signal<T>*, sont gérés indépendamment par les *threads* des nœuds. Le *thread* d'un nœud est responsable d'éjecter ou d'injecter des paquets dans le cas échéant. Le type *sc_signal<T>* est employé afin de simuler les déplacements des paquets dans le modèle au cycle près.

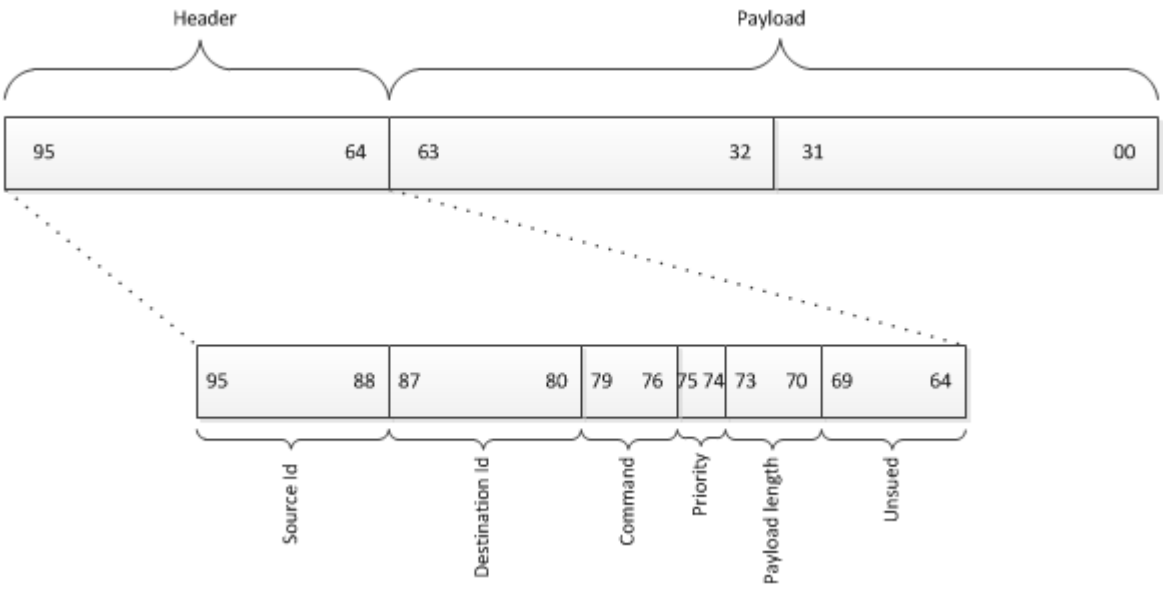

Figure 3-4 : Paquet RoC

Un paquet RoC, appelé *RoCTransaction*, est une structure ayant 96 bits dont 32 bits sont réservés pour l'entête et 64 bits pour les données utiles. La Figure 3-4 représente la structure d'un tel paquet. Tel que mentionné préalablement, l'ajout d'un nouveau réseau sur puce doit définir son propre type de paquet.

Dans l'entête, le « Source Id » est l'identifiant du module source tandis que le « Destination Id » est l'identifiant du module destinataire encodé sur 8 bits. La commande, encodée sur 4 bits, est la commande du paquet. Voici la liste des commandes possible :

- NOC_TEST
- NOC_CONFIG
- NOC_WRITE
- NOC_WRITE_OFFSET
- NOC_READ
- NOC_RESPOND

Les deux premières commandes sont des commandes utiles à la reconfiguration du RoC ainsi qu'à l'injection de vecteur de test dans le modèle. Ces commandes ne sont pas utiles dans le cadre de ce mémoire. Le restant des commandes sont les commandes utilisées lors du fonctionnement général d'une application. Dans le cas d'une commande avec offset (NOC_WRITE_OFFSET, NOC_READ et NOC_RESPOND), 32 bits du payload sont utilisés afin de transmettre l'offset. C'est la responsabilité des adaptateurs de supporter correctement les commandes.

La priorité, encodée sur 2 bits, représente la priorité du paquet. Le modèle utilise ce champ afin de déterminer dans quelle fifo d'entrée le paquet doit aller. Le « Payload length », encodé sur 4 bits, représente le nombre d'octets de donnée utile présent dans le paquet. Par conséquent, le nombre maximal d'octets de donnée utile est de huit octets. Finalement, il reste 6 bits non utilisés pouvant être utiles lors de développements futurs.

3.4.1.2 Bas niveau

Le modèle bas niveau du Rotator-On-Chip sera utilisé lors de l'implémentation sur puce FPGA. Plus précisément, ceci aura lieu lors du raffinement de notre architecture haut niveau vers une cible FPGA. Le modèle bas niveau doit incorporer les mêmes fonctionnalités (bidirectionnel, *backpressure*, etc.) et il doit avoir la même interface que le modèle haut niveau.

Pour réaliser le modèle bas niveau, nous avons besoin d'un langage de description matériel. VHDL [6] et Verilog [7] sont deux exemples de langages de description matériel, mais puisque notre expérience se situe avec le langage VHDL, nous allons utiliser ce dernier afin de réalisé le modèle bas niveau du Rotator-On-Chip. Le modèle ainsi réalisé devra être synthétisable, avoir les mêmes caractéristiques que le modèle haut niveau et conserver les délais observés dans [3].

```
entity roc is
		generic (
			NB_PRIORITY		: integer := 1;		-- Must be greater than 0
			NB_NODES		: integer := 6;		-- Must be greater than 0
			NB_CHANNELS		: integer := 2;		-- Must be greater than 0
			BACK_PRESSURE		: integer := 15;	-- Must be greater than 0
			DATA_PATH		: integer := 96
		);
		port (
			-- Clock, reset
			Clk		: in  std_logic;
			Reset		: in  std_logic;

			-- RoC slave mirrored
			ROC_S_DATA	: out std_logic_vector(0 to (NB_NODES*DATA_PATH)-1);
			ROC_S_EMPTY	: out std_logic_vector(0 to NB_NODES-1);
			ROC_S_RE	: in std_logic_vector(0 to NB_NODES-1);

			-- RoC master mirrored
			ROC_M_DATA	: in std_logic_vector(0 to (NB_NODES*DATA_PATH)-1);
			ROC_M_FULL	: out std_logic_vector(0 to (NB_NODES*NB_PRIORITY)-1);
			ROC_M_WE	: in std_logic_vector(0 to (NB_NODES*NB_PRIORITY)-1)
		);
end roc;
```

Figure 3-5: Modèle bas niveau du Rotator-On-Chip

À l'aide de la Figure 3-5 , il est possible d'observer que les génériques permettent de configurer le modèle bas niveau à souhait tout comme le modèle haut niveau. On remarque aussi que le branchement des adaptateurs doit se faire à l'intérieur du vecteur ayant comme préfixe *ROC_*.

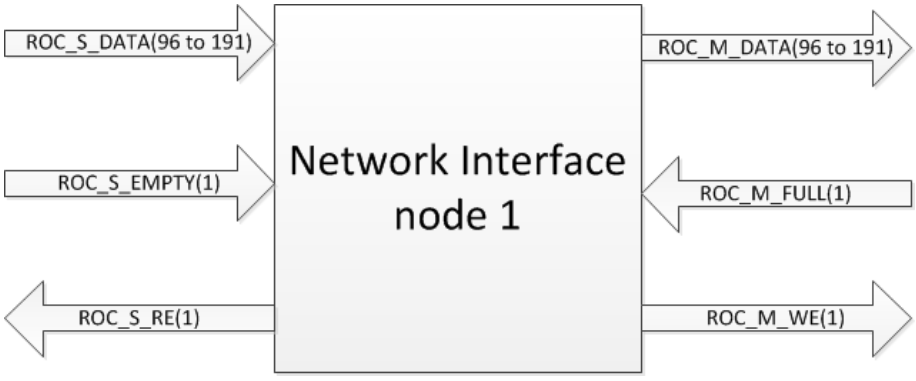

Figure 3-6 : Exemple d'un adaptateur se branchant au nœud 1

La Figure 3-6 montre un exemple d'un adaptateur réseau se branchant au nœud 1 du RoC en utilisant la définition présenté par la Figure 3-5.

```
entity table is
port
(
		SpaceID		: in  std_logic_vector(0 to 7);
		NodeID		: out std_logic_vector(0 to 7)
);
end entity table;

architecture behaviour of table is
begin
	SpaceID2NodeID : process (SpaceID)
	begin
		case SpaceID is
		when X"12" =>
			NodeID <= X"00";
		when X"13" =>
			NodeID <= X"01";
		when X"14" =>
			NodeID <= X"02";
		when X"15" =>
			NodeID <= X"02";
		when X"16" =>
			NodeID <= X"03";
		when X"17" =>
			NodeID <= X"04";
		when others =>
			NodeID <= X"00";
		end case;
	end process SpaceID2NodeID;
end architecture behaviour;
```

Figure 3-7 : Exemple d'assignation à bas niveau

Le fichier haut niveau d'assignation est remplacé par un fichier VHDL indiquant la même information. La Figure 3-7 montre un exemple d'assignation pour une configuration donnée. En plus de représenté l'assignation, cette entité remplace la méthode *FindNodeId* provenant de

NoCUtilityIF du modèle haut niveau. Par conséquent, ce fichier représente le raffinement de l'interface *NoCIFUtility*.

Le modèle bas niveau utilise la même structure de paquet RoC défini lors de l'implémentation haut niveau du modèle. En conservant la même structure de paquet, aucun changement ne doit être apporté aux interfaces de communication lors du raffinement.

3.4.2 Développement des adaptateurs

3.4.2.1 Haut niveau

Un adaptateur est un dispositif permettant à deux équipements différents de fonctionner ensemble. Le meilleur exemple est les adaptateurs pour les prises électriques entre le standard nord-américain et ceux de l'Europe. Dans les systèmes embarqués, les adaptateurs servent exactement à la même tâche. Ils permettent d'adapter des communications d'un certain type vers un autre type de communication. Par le fait même, les adaptateurs facilitent la réutilisation des blocs fonctionnels. Par exemple, un bloc fonctionnel utilisant une interface de type OCP peut être réutilisé dans un système de bus AMBA en utilisant un adaptateur effectuant une transition du protocole OCP vers un protocole de bus AMBA.

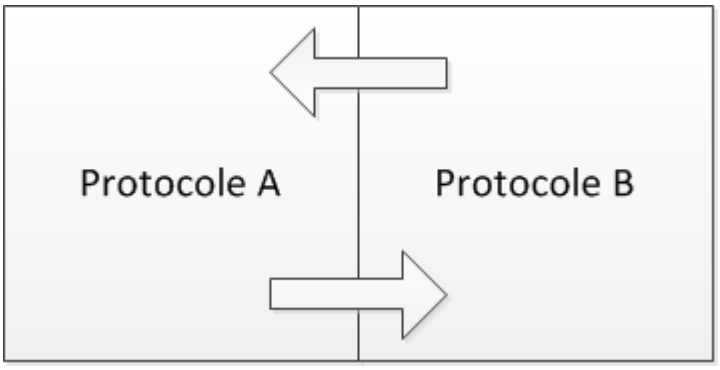

Figure 3-8 : Adaptateur traditionnel

La façon traditionnelle de concevoir un adaptateur est représentée par la Figure 3-8 où on adapte un protocole A vers B et vice-versa. Le problème avec une telle approche survient

lorsqu'on introduit un protocole C. Si l'on désire effectuer un adaptateur de A vers C, il faut refaire la majorité du travail, car l'interface entre A et B diffère de celle entre A et C. Ceci est également vrai lorsqu'on désire faire un adaptateur entre B et C.

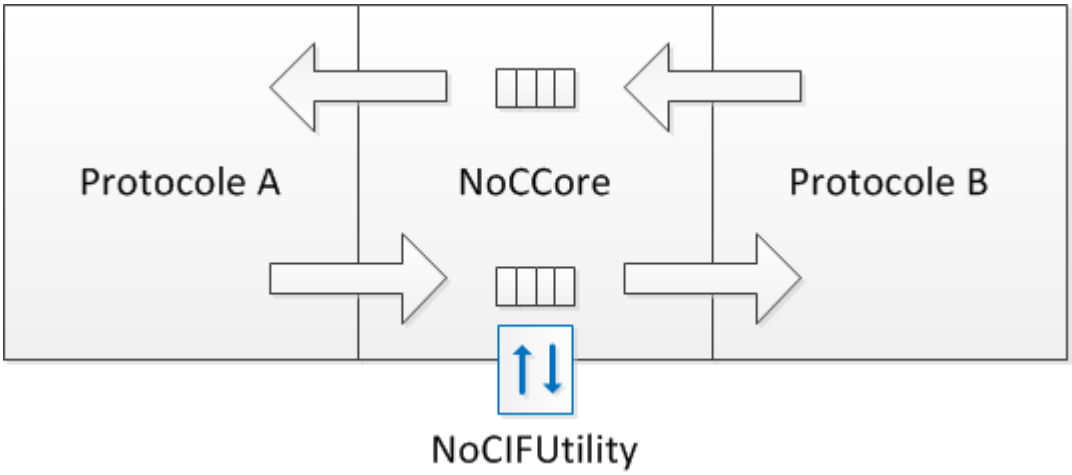

Figure 3-9 : Adaptateur générique

Pour remédier à ce problème, nous avons introduit un adaptateur commun de type *NoCCore* utilisant une interface *NoCInterface*. Cet adaptateur noyau a comme caractéristique d'être générique à l'ensemble des réseaux sur puce et permet la réutilisation unitaire des adaptateurs. La Figure 3-9 présent un adaptateur générique où l'on remarque l'introduction de l'adaptateur noyau appelé *NoCCore* et de deux adaptateurs unitaires (noté Protocole A et Protocole B).

Toujours à l'aide de la Figure 3-9, l'adaptateur unitaire de gauche effectue une adaptation du protocole A vers le protocole *NoCInterface* tandis que l'adaptateur unitaire de droite effectue une adaptation du protocole B vers le protocole *NoCInterface*. Lorsqu'on introduit un nouveau protocole C, il suffit d'effectue l'adaptateur unitaire C vers le protocole *NoCInterface*. Une fois cet adaptateur réalisé, la conversion entre A vers C et de B vers C est automatique [17]. Ceci s'apparente à l'utilisation de protocoles standards (i.e. OCP et VCI) mais offre un support externe pour interroger l'architecture de réseau sur puce.

Le noyau *NoCCore* détient deux fifos ayant chacun une profondeur de un. Les fifos sont accédées via l'interface *NoCInterface* ayant les méthodes *push, pop, FindNodeId* et *GetNodeId.* Les éléments ajoutés et retirés des fifos ont la structure de type *NoCTransaction* et sont génériques à l'ensemble des réseaux sur puce. Cette structure contient l'identifiant du module source, l'identifiant du module destination, la commande, un pointeur vers les données utiles, le nombre d'octets de données utiles et l'offset de l'accès (si la commande supporte un offset). Chaque adaptateur est responsable de la traduction des données reçues sur son interface de protocole spécifique vers une représentation *NoCTransaction* permettant ainsi l'interopérabilité.

En plus des méthodes *push* et *pop*, l'interface *NoCInterface* permet l'utilisation de la méthode *FindNodeId* et *GetNodeId*. La méthode *FindNodeId* retourne l'identifiant du nœud auquel un module est branché. Pour retourner cette information, l'interface *NoCInterface* utilise le port *NoCIFUtility* provenant du *NoCCore*. Puisque le port *NoCIFUtility* est branché avec le NoC, c'est en réalité le modèle du réseau sur puce qui retourne cette information à l'aide de son fichier d'assignation vue précédemment. La méthode *GetNodeId* retourne l'identifiant du nœud auquel l'adaptateur courant est branché. Cette information est obtenue lors de l'instanciation de l'adaptateur.

Nous avons réalisé deux types d'adaptateur unitaire. Le premier est un adaptateur FSL « Fast Simplex Link » et le second est un adaptateur RoC. Si l'on intègre un nouveau réseau sur puce, par exemple un modèle de maillage ayant une interface de type MESH, et que l'on réalise l'adaptateur unitaire MESH, et bien, il sera possible de connecter un FSL vers une MESH et même un RoC vers une MESH!

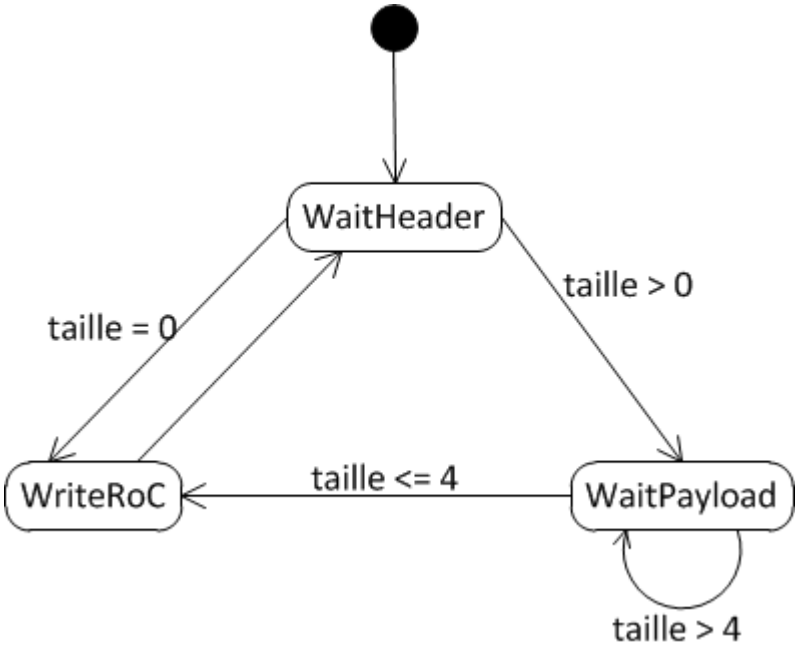

Figure 3-10 : Machine à état pour FSLRoCAdapter

La combinaison des adaptateurs unitaires RoC et FSL forment l'adaptateur *FSLRoCAdapter*. Essentiellement, cet adaptateur effectue la traduction des communications FSL vers des communications RoC et vice-versa. Puisqu'une communication FSL est sur 32 bits, il faut tout au plus trois communications FSL pour produire une communication RoC de 96 bits. Le travail effectué par cet adaptateur est illustré par la Figure 3-10 montrant la machine à état d'une communication entrant dans le RoC (FSL vers RoC).

Le premier état, appelé *WaitHeader*, consiste à la réception de l'entête ainsi que le décodage de la taille des données utiles à recevoir. Si la taille des données donnée utile est égale à zéro, aucune autre réception n'est attendue et on passe à l'état *WriteRoC*. Dans le cas contraire, on passe au deuxième état, appelé *WaitPayload*. L'entête reçu est ajouté au paquet RoC en cours de préparation.

Le deuxième état, appelé *WaitPayload*, attend la réception des données utiles. Lorsque la réception est effectuée, on vérifie la taille des données utiles décodées lors du premier état. Si la

taille des données utiles est inférieure ou égale à quatre, on passe à l'état *WriteRoC*. Dans le cas contraire, on décrémente la taille de quatre et on reste à l'état courant. Dans tous les cas, les données utiles reçues sont ajoutées au paquet RoC en cours de préparation.

Le dernier état, appelé *WriteRoC*, consiste à écrire le paquet RoC dûment formé lors des états précédents dans le RoC. Lorsque cette opération est effectuée, on retourne au premier état et on initialise la variable du paquet RoC en cours de préparation.

De manière similaire, l'adaptateur *FSLRoCAdapter* possède une machine à état effectuant une communication sortant du RoC (RoC vers FSL).

3.4.2.2 Bas niveau

Une des tâches effectuées lors du raffinement est de remplacer les blocs haut niveau par les blocs similaires au niveau RTL (bas niveau). Les adaptateurs n'échappent pas à un tel raffinement. Par conséquent, il faut proposer un adaptateur bas niveau qui raffinement l'adaptateur haut niveau *FSLRoCAdapter*. Sans grande surprise, l'adaptateur bas niveau, appelé également *FSLRoCAdapter*, est l'implémentation RTL identique de l'adaptateur haut niveau.

```
        entity fslrocadapter is
                generic (
                        NB_PRIORITY    : integer := 1; -- Must be greater then 0
                        DATA_PATH      : integer := 96 -- RoC data path
                );
                port (
                -- Clock, reset
                Clk                    : in  std_logic;
                Reset                  : in  std_logic;

                -- FSL slave (data coming from microblaze)
                FSL_S_READ     : out std_logic;
                FSL_S_DATA     : in std_logic_vector(0 to 31);
                FSL_S_EXISTS   : in std_logic;

                -- FSL master (data coming from RoC)
                FSL_M_WRITE    : out std_logic;
                FSL_M_DATA     : out std_logic_vector(0 to 31);
                FSL_M_FULL     : in std_logic;

                -- RoC slave
                ROC_S_DATA    : in std_logic_vector(0 to DATA_PATH-1);
                ROC_S_EMPTY   : in std_logic;
                ROC_S_RE      : out std_logic; -- Read enable

                -- RoC master
                ROC_M_DATA    : out std_logic_vector(0 to DATA_PATH-1);
                ROC_M_FULL    : in std_logic_vector(0 to NB_PRIORITY-1);
                ROC_M_WE      : out std_logic_vector(0 to NB_PRIORITY-1) -- Write enable
                );
end fslrocadapter;
```

Figure 3-11 : Entité de l'adaptateur FSL-RoC

La Figure 3-11 représente l'entité de l'adaptateur *FSLRoCAdapter* ayant une interface FSL maître-esclave et une interface RoC maître-esclave. Cet adaptateur est le raffinement de l'adaptateur haut niveau *FSLRoCAdapter* vu dans la section précédente.

Toujours à l'aide de la Figure 3-11, on remarque que les ports ayant le préfixe *ROC_* se branchent dans les ports du modèle bas niveau du RoC vu à la Figure 3-5. Les interfaces FSL de l'adaptateur se branchent dans le processeur auquel le nœud est rattaché tandis que les interfaces RoC maître et esclave se branchent à l'index *X* dans le modèle bas niveau du Rotator-On-Chip où *X* indique le numéro du nœud.

Le travail effectué par cet adaptateur est le même que celui du modèle haut niveau. Le détail de la machine à état ainsi que son explication a été présentée lors de la section précédente.

3.4.3 Application et assignation sur l'architecture ciblée

Pour valider notre réseau sur puce, nous avons développé un système complet de traitement d'images. Nous assignerons (*mapping*) cette application à une architecture et à l'aide de cette architecture, nous serons en mesure de déterminer le bon fonctionnement de notre modèle de réseau sur puce et également d'obtenir des métriques de performances de notre système. Les résultats obtenus par le biais de la plate-forme de test seront présentés dans le chapitre suivant. Dans ce qui suit, nous présentons d'abord l'application, puis les justifications et les détails de l'assignation.

3.4.3.1 Application utilisée

L'application utilisée dans le cadre de notre mémoire est le Motion JPEG (MJPEG) qui est le décodage d'un flot d'images JPEG [20].

Le code de départ de l'algorithme MJPEG a été développée par le groupe d'Architecture des Systèmes Intégrés et Microélectronique (ASIM) du Laboratoire d'Informatique de Paris 6 (LIP6) de l'Université Pierre et Marie [45].

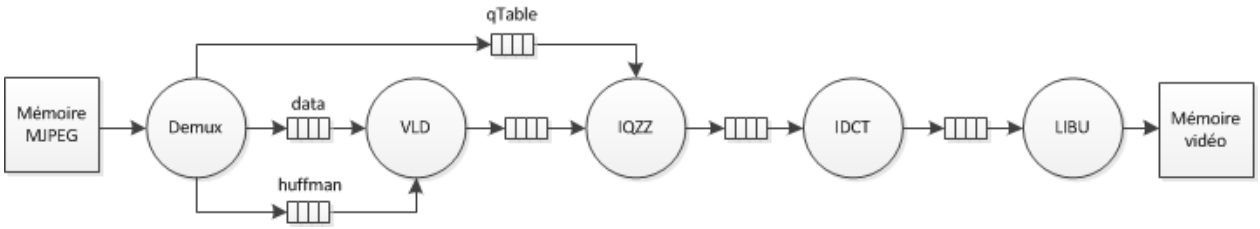

Figure 3-12: MJPEG

La Figure 3-12 montre le fonctionnement de l'application MJPEG. La figure montre le flot de données d'une telle application. Voyons en détail la fonction de cette application.

1. Tout d'abord, le fichier MJPEG est emmagasiné dans la mémoire MJPEG. Ce fichier doit contenir les marqueurs nécessaires au décodage par le module *Demux*. Dans le cadre de ce mémoire, nous allons nous concentrer sur une animation MJPEG ayant 48 pixels par 48 pixels. L'animation s'agit d'un petit avion qui effectue une rotation dans le sens antihoraire d'une montre.

2. Le module *Demux* « Démultiplexage du flux de données » se charge de lire la mémoire et d'analyser les marqueurs qui délimitent les sections du fichier. Il envoie la(les) table(s) de déquantification (qTable) vers le module IQZZ, puis les tables d'Huffman ainsi que les données utiles vers *VLD*.

3. Le module *VLD* « Décompression du flux compressé » décode les données utiles avec les tables d'Huffman reçu préalablement puis envoie les valeurs des symboles trouvées vers IQZZ.

4. Le module *IQZZ* « Application de facteurs de quantification et transformé ZigZag » effectue un travail de déquantification et de parcours en dents de scie (en anglais « unzig-zag »), puis achemine les données à *IDCT*.

5. Le module *IDCT* « Transformée cosinus inverse » effectue une « inverse discrete cosine transform » avant d'envoyer les données décodées au module *LIBU*.

6. Le module *LIBU* « Construction de lignes complétes d'image » transforme les blocs en ligne et envoie les pixels vers une mémoire vidéo.

Le projet initial nécessitait la plate-forme de développement et d'exploration architecturale nommée DSX. Nous avons donc modifié le code initial afin de remplacer les appels aux fonctions de communication de la plate-forme DSX par des appels aux fonctions de communication SPACE. Par la migration du code initial, le nouveau code source tourne rondement dans l'outil SPACE.

3.4.3.2 Assignation de l'application sur une architecture

L'objectif de cette partie du travail est d'abord de valider la fonctionnalité du réseau de communication et de ses adaptateurs. Nous avons opté pour une solution multiprocesseur 100% logiciel, où chaque processeur est branché directement au réseau sur puce. Également, le fait d'être complètement logiciel, cette solution rencontre les besoins actuel du marché de l'embarqué, qui est de pouvoir accélérer le développement d'application tout en facilitant la réutilisation et les mises à jour ultérieur du système. Certes, l'exploration d'architectures incluant des accélérateurs matériels permettrait de maximiser davantage les performances du système tout en minimisant la consommation, mais cela fera partie des travaux futurs.

La cible finale est une carte FPGA Xilinx proposant le processeur « soft-core » appelé µBlaze. Notre solution multiprocesseur sera donc composée d'un ensemble de µBlaze. Ce dernier possède des liens directs appelés FSL qui seront branchés directement sur les adaptateurs dument

développés. Nous avons décidé d'utiliser les liens directs pour brancher les processeurs avec le réseau sur puce principalement pour deux raisons. Premièrement, le développement ainsi que la validation de composante écrite dans un langage de description matériel sont un processus fastidieux. Il est possible de brancher le processeur avec le Rotator-On-Chip par le biais du bus OPB, mais, il est plus simple et facile de développer un adaptateur FSL vers RoC qu'un adaptateur OPB vers RoC au niveau RTL. Deuxièmement, nous croyons qu'un branchement direct du processeur avec le réseau sur puce présentera de meilleures performances comparativement à un branchement par le biais d'un bus local assujetti à un arbitrage.

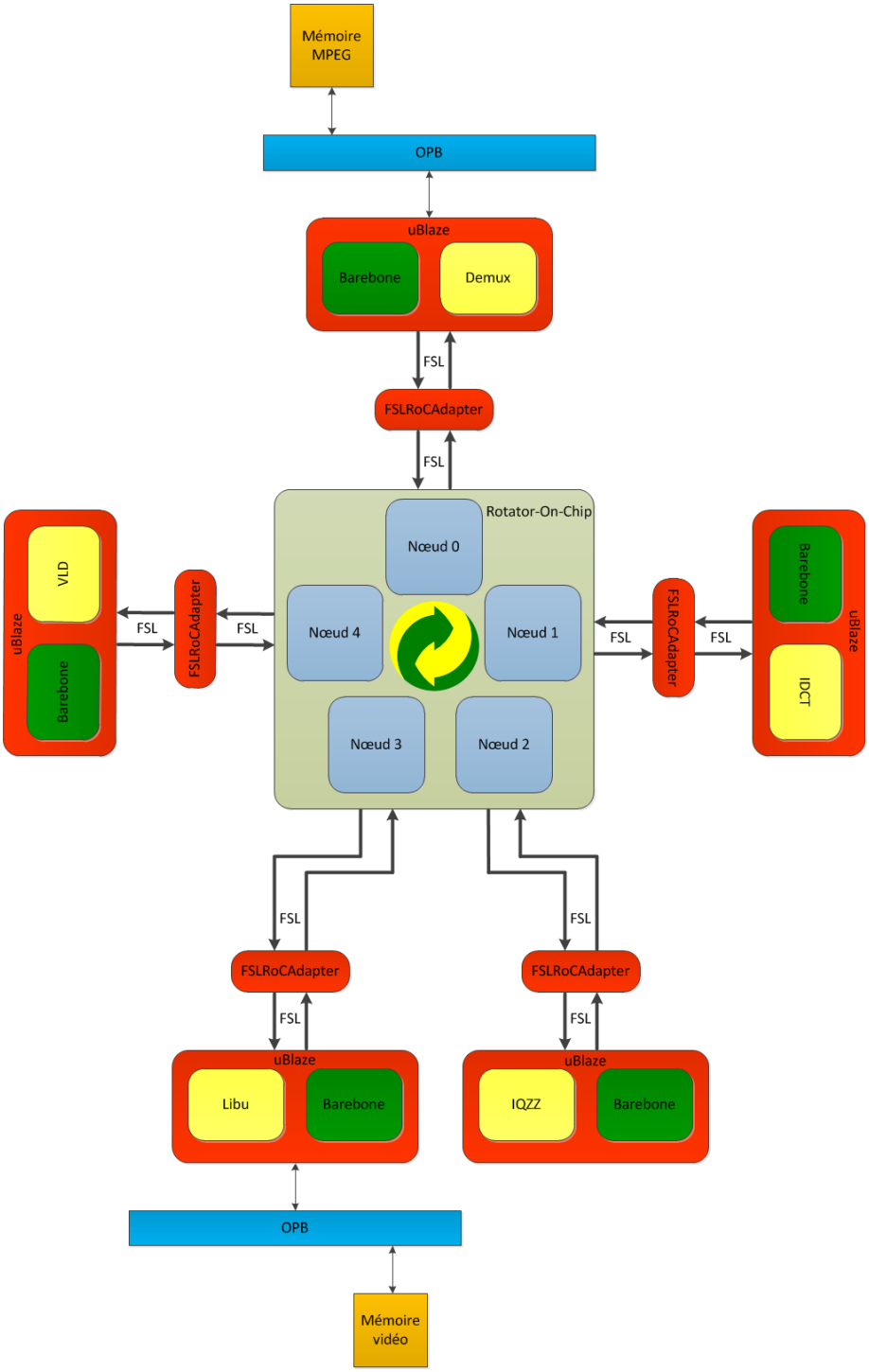

Figure 3-13 : Résultat de l'assignation

Notre architecture, présentée à la Figure 3-13 , propose cinq processeurs branchés sur le réseau sur puce par l'intermédiaire d'un adaptateur appelé *FSLRoCAdapter*. Chaque processeur

possède sa mémoire d'instruction et de donnée (non représenté sur la figure). Par ailleurs, les processeurs ayant la tâche *Demux* et *Libu* possèdent un bus local. Ce bus est utilisé afin d'accéder leur mémoire respective (la mémoire MJPEG et la mémoire vidéo). Le tout est branché à une horloge ayant une fréquence de 100 MHz.

Dans l'application du MJPEG, une tâche peut communiquer avec plusieurs autres tâches. C'est le cas de la tâche *IQZZ* qui reçoit des informations de *VLD* et *Demux.* Étant donné qu'un seul lien de communication relie le processeur avec le réseau sur puce (lien FSL), c'est au système d'exploitation de la tâche réceptrice, par exemple *IQZZ,* de différencier les sources et de placer les données à un endroit approprié pour une lecture future. Cette opération est transparente pour la tâche logicielle exécutée, car le tout est effectué en coulisse par le système d'exploitation. La section suivante traitera de cette approche.

3.4.4 Développement du logiciel embarqué

L'outil SPACE offre des primitives de communication de base. Ces primitives de communication sont reconnaissables par les fonctions *ModuleRead, ModuleWrite, DeviceRead* et *DeviceWrite.* L'utilisation de ses primitives de communication nécessite un bus local afin de réaliser la communication. Concrètement, les appels aux fonctions de communication effectuée par un module situé sur un processeur nécessitent la suite des appels suivant :

1. Appel aux primitives de communication de base à partir du code utilisateur
2. Traitement propriétaire effectué par l'outil SPACE
    a. Appel aux fonctions du système d'exploitation
    b. Appel aux fonctions d'abstractions matérielles

Le travail effectué par le point 2 est principalement une tâche de contrôle réalisant l'opération de communication. Par exemple, dans le cas d'un appel à la fonction *ModuleRead,* le point 2 doit, dans un premier temps, interroger la fifo logicielle par le biais des fonctions du système d'exploitation afin de déterminer si les données utiles sont présentes. Si les données s'y

trouvent, les données sont retournées. Dans le cas où les données ne sont pas présentes, le point 2 effectue les appels de communication bas niveau sur le bus via les fonctions d'abstraction matérielles.

Tel qu'indiqué, le traitement propriétaire effectué par l'outil SPACE doit obligatoirement utiliser un bus local. Or, tel que présenté lors de la section précédente, l'architecture développée indique que les processeurs sont directement branchés sur le réseau sur puce par le biais des liens FSL. Par conséquent, les appels aux fonctions d'abstraction matérielle ne sont plus valides et nous devons fournir notre propre logique afin de supporter l'utilisation des liens FSL. Ceci est facilement réalisable par l'extension des fonctions d'abstraction matérielle déjà présente.

De plus, l'outil SPACE fournis deux systèmes d'exploitation : µC et Unity. µC est un système exploitation développé par Micrium [23] ayant comme caractéristique de supporter plusieurs tâches. À l'opposé, Unity est un système d'exploitation développé par l'équipe de SPACE qui supporte une seule tâche. Puisque nous allons incorporer une tâche par processeur, l'utilisation de Unity semble un bon choix. Lors des essais préliminaires, nous avons remarqué que la taille du logiciel embarqué ayant Unity comme système d'exploitation était énorme. En effet, chaque binaire avait environ une taille de 130 kilooctets (ceci comprend le système d'exploitation et la tâche). Ceci est énorme lorsqu'on souhaite intégrer le tout sur puce FPGA. L'explication de la taille impressionnante du fichier binaire est que le système d'exploitation incorpore un large éventail de fonctionnalités non utilisé.

Dans l'optique de minimiser l'empreinte mémoire du code logiciel embarqué, nous avons créé un système d'exploitation minimaliste supportant une seule tâche ainsi que les fonctionnalités requises. Les seules fonctionnalités requises sont les opérations sur les fifos (création d'une fifo logicielle, lecture dans une fifo logicielle et écriture dans une fifo logicielle). Le nouveau système d'exploitation, appelée « Barebone » supporte une seule tâche et fournis uniquement des primitives pour la lecture et l'écriture dans des fifos logicielles. Ceci nous permet de réduire considérablement la taille de chaque binaire à environ 30 kilooctets. Une économie de 100 kilooctets par processeur!

3.4.4.1 Virtualisation VGA

La virtualisation VGA consiste à concevoir un modèle de contrôleur VGA dans l'environnement haut niveau afin d'afficher le contenu de la mémoire vidéo sur l'écran du concepteur. La principale utilisée d'une virtualisation VGA est de détermine si le décodage s'effectue correctement. Sans un tel affichage, il peut être difficile de valider le décodage des images, car on doit comparer octet par octet avec le modèle de référence. Une autre utilité du contrôleur VGA est qu'il montre si le temps réel a été atteint ou non. Dans le cas où le temps réel n'a pas été atteint, l'affichage montera une mise à jour progressive de l'image, exactement comme on le verrait lors d'une implémentation réelle sur puce. Dans le cas où le temps réel est atteint, on verra chaque image s'afficher correctement les unes après les autres. Finalement, ayant une application telle que le MJPEG, il est d'autant plus intéressant pour le concepteur de système de voir le résultat du décodage défilé sous ses yeux avides.

Dans l'environnement SPACE, il est possible de créer un tel composant par le biais de périphérique. Dans la terminologie de SPACE, un périphérique est un esclave sur le bus répondant aux requêtes des maîtres. Chaque périphérique peut être accédé en écriture ou lecture dépendamment du type de requête. De plus, le maitre peut, s'il le désire, fournir un « offset » lors de l'accès permettant d'encoder de l'information supplémentaire.

Soucieux de la réutilisation du composant de base VGA, nous avons créé une classe de service nommé *BasicVGAController* fournissant la logique nécessaire pour la création de la virtualisation VGA. Le *BasicVGAController* est responsable de la création de la vue OpenGL, de l'affichage de l'écran et de la synchronisation de la fenêtre virtuelle. Cette classe de service affiche un tableau de pixels au format RGA (Red-Grend-Blue-Alpha). Par conséquent, si le périphérique utilisant le *BasicVGAController* reçoit autre chose que des pixels au format RGBA, il devra effectuer un traitement sur ses pixels avant l'envoi du tableau de pixel au format RGBA vers cette classe.

Le périphérique SPACE nommé *VGAController* doit effectuer deux tâches. La première tâche est de traiter les données reçues de la simulation et de les sauvegarder dans sa mémoire vidéo. Sa mémoire vidéo peut être du type qu'il le désire. La seconde tâche, s'effectuant à toutes les $^{1}/_{taux\ de\ rafraichement}$, est de faire une capture de la mémoire vidéo et d'afficher cette capture à la fenêtre virtuelle par le biais du *BasicController* au format RGBA.

Chaque périphérique SPACE hérite des méthodes *SlaveSpecificRead* et *SlaveSpecificWrite* servant respectivement aux requêtes de lecture et d'écriture. Un accès en écriture au périphérique VGA sert pour enregistrer les valeurs des pixels tandis qu'un accès en lecture au périphérique sert au module communicant avec le contrôleur VGA d'indiquer si l'image a été affichée et que la mémoire vidéo peut être réécrite.

Lorsque la simulation se termine, le contrôleur VGA affiche en rafale les captures de la mémoire vidéo. Si le temps réel a été atteint par la simulation, l'affiche rendu par le contrôleur VGA sera en temps réel. Dans le cas contraire, l'affiche vidéo sera saccadée. Ceci permet donc au concepteur de rapidement et facile voir s'il a attend le temps réel. Dans l'objectif d'une métrique précise, en utilisant la référence de temps réel étant de 30 images par seconde, on utilise le temps de simulation et on effectue une règle de trois avec le nombre d'images décodées. Le temps réel est atteint lorsqu'on franchit le cap des 30 images par seconde.

3.4.5 Raffinement logiciel/matériel vers une cible FPGA

Le raffinement vers une cible FPGA est le processus par lequel une architecture haut niveau est raffinée vers une implémentation sur puce FPGA. L'outil SPACE, par le biais de *GenX*, permet le raffinement d'architectures vers une cible FPGA. Cependant, une architecture incluant un réseau sur puce n'est pas supportée. La méthodologie de raffinement, tel que proposé par l'outil SPACE, permet un raffinement automatique avec un minimum d'interventions de la part de l'usager.

Dans [12], Gabriel Oshiro, un membre de notre groupe de recherche, travaille actuellement sur la problématique d'effectuer le raffinement automatique d'une architecture incluant un réseau sur puce vers une cible FPGA. Sans entrer dans les détails, le raffinement automatique, effectué par l'outil SPACE et décrit dans [12] doit effectuer le travail suivant :

- Création automatique du fichier d'assignation
- Instanciation des composantes bas niveaux (adaptateurs, réseau sur puce, etc.)
- Branchement des composantes
- Création du projet de sortie pour le logiciel tiers désiré (i.e. EDK de Xilinx)

Dans le cadre de ce mémoire, nous avons effectué manuellement les points mentionnés ci-dessus. Ceci nous a permis de valider le fonctionnement de notre architecture à bas niveau sans le recours au raffinement automatique tel que proposé par M. Oshiro. Une fois le travail de ce dernier terminé, il sera intégré à notre méthodologie dans un travail ultérieur.

CHAPITRE 4 RÉSULTATS ET DISCUSSION

Ce chapitre présente les résultats obtenus lors de notre expérimentation. Plus précisément, nous allons présenter trois catégories de résultats. Premièrement, nous allons démontrer le bon fonctionnement du modèle haut niveau et bas niveau à l'aide d'un banc d'essai dûment réalisé. Deuxièmement, nous allons montrer que le comportement du modèle haut niveau correspond au modèle bas niveau et finalement, nous allons montrer les résultats de notre architecture développée exécutant l'application MJPEG.

4.1 Banc d'essai

Pour nous assurer du bon fonctionnement de nos modèles (haut et bas niveau) de réseaux sur puce développés, nous devons les valider avec un banc d'essai afin de nous assurer que les modèles soient bien conçus. Concrètement, cette validation vise à déterminer si les paquets naviguent correctement dans le modèle. Si des paquets sont perdus ou altérer, le banc d'essai sera en mesure de le confirmer.

4.1.1 Haut niveau

Le banc d'essai haut niveau est écrit dans le langage C/C++ et utilise la libraire de simulation SystemC. Dans le banc d'essai, les modules sont des blocs matériels et sont directement branchés aux interfaces du Rotator-On-Chip sans le recours aux adaptateurs. Il y a autant de modules qu'il y a de nœuds. Le banc d'essai développé est plutôt simpliste et consiste à l'envoi de paquets à des destinateurs aléatoires outre lui-même. Le banc d'essai est facilement paramétrable par les directives *#DEFINE* suivantes :

- FIFO_IN_SIZE : Profondeur des fifos d'entrées
- FIFO_OUT_SIZE : Profondeur de la fifo de sortie
- NB_CHANNELS : Nombre de canaux
- NB_NODES : Nombre de noeuds
- NB_PRIORITY : Nombre de priorités

- BACKPRESSURE : Valeur du backpressure (dois être égale ou inférieure à la valeur de la profondeur de fifo de sortie)
- NB_PACKET_TO_SEND : Nombre de paquets que chaque nœud doit émettre

Le banc d'essai se charge de la création du fichier d'assignation utile pour le Rotator-On-Chip selon les directives fournies et exécute la simulation. Lors de l'exécution du banc d'essai, les modules émettent des paquets ayant comme donnée utile le numéro du paquet. Le numéro du paquet est en fait un nombre incrémental pour chaque destinataire débutant à 0. Lors de la réception des paquets, chaque module s'assure que les paquets sont dans l'ordre. Si un module reçoit un paquet ayant comme donnée « 10 » et que le paquet suivant, il reçoit un paquet ayant comme donnée « 12 », ceci veut dire qu'un paquet a été perdu. Par conséquent, si les paquets reçus ne sont pas dans l'ordre ou pire, que les données reçues ne représentent pas un numéro de paquet, ceci indique que le modèle présente des anomalies devant être corrigées. Par ailleurs, la détection des altérations des paquets relève du banc d'essai et non de l'engin d'analyse.

Lorsque la simulation du banc d'essai se termine, le banc d'essai affiche le nombre de paquets reçus et perdus pour chaque émetteur. En fait, le banc d'essai présente les mêmes métriques que l'engin d'analyse mais calculé par le banc d'essai. Par conséquent, il est possible de comparer ces données avec l'engin d'analyse. Si les résultats concordent, le modèle haut niveau est considéré comme fonctionnel. Par ricochet, on valide l'engin d'analyse. Nous utilisons le terme « considéré », car des anomalies peuvent toujours être présentes dans le modèle qui n'ont pas ressorti lors de cet exercice, car notre banc d'essaie de couvre pas tous les cas.

```
Starting simulation.
Module 0 sent 0 packets to module 0
Module 0 sent 184 packets to module 1
Module 0 sent 214 packets to module 2
Module 0 sent 205 packets to module 3
Module 0 sent 197 packets to module 4
Module 0 sent 200 packets to module 5
Module 0 has read : 954
Module 0 lost 0 packets from module 0
Module 0 lost 0 packets from module 1
Module 0 lost 0 packets from module 2
Module 0 lost 0 packets from module 3
Module 0 lost 0 packets from module 4
Module 0 lost 0 packets from module 5
```

```
       Module 1 sent 188 packets to module 0
       Module 2 sent 194 packets to module 0
       Module 3 sent 192 packets to module 0
       Module 4 sent 190 packets to module 0
       Module 5 sent 190 packets to module 0

       Simulation has ended @20990 ns
Simulation wall clock time: 0 seconds.
```

Figure 4-1 : Aperçu des résultats lors de l'exécution du banc d'essai

La Figure 4-1 présente un aperçu des résultats obtenus par le banc d'essai lors de l'exécution de ce dernier pour les valeurs de directives suivantes :

- FIFO_IN_SIZE : 16
- FIFO_OUT_SIZE : 16
- NB_CHANNELS : 2
- NB_NODES : 6
- NB_PRIORITY : 1
- BACKPRESSURE : 16
- NB_PACKET_TO_SEND : 1000

Il n'est pas possible ici de présenter la totalité des résultats obtenus par le banc d'essai, car le tout tiendrait sur plusieurs pages. Par conséquent, nous présentons uniquement les résultats pertinents au module 0 qui est branché au nœud 0 du Rotator-On-Chip. Bien que ceci soit un aperçu, il y a suffisamment d'information pour valider le modèle.

```
       [+]Data for node   0

        |- Number of read                          :     954

        |- Number of write                         :    1000

        |- Number of packet refused                :       0

        |

        |-[+] Clockwise

        |  |-[+] Fifo IN
```

```
|  |  |- Depth : 16
|  |  |-[+] QoS 0
|  |  |  |- Largest utilization : 16 (100%)
|  |  |  |- Average utilization :  8 (50%)
|  |-[+] Fifo OUT
|  |  |- Depth : 16
|  |  |-[+] QoS 0
|  |  |  |- Largest utilization : 15 (93%)
|  |  |  |- Average utilization :  7 (43%)
|
|-[+] Counter clockwise
|  |-[+] Fifo IN
|  |  |- Depth : 16
|  |  |-[+] QoS 0
|  |  |  |- Largest utilization : 16 (100%)
|  |  |  |- Average utilization :  8 (50%)
|  |-[+] Fifo OUT
|  |  |- Depth : 16
|  |  |-[+] QoS 0
|  |  |  |- Largest utilization : 15 (93%)
|  |  |  |- Average utilization :  7 (43%)
|
|-[+] Communication details
|  |-[+] Node 0
|  |  |-[+] QoS 0
|  |  |  |- Minimum write latency : 0 cycles
|  |  |  |- Maximum write latency : 0 cycles
|  |  |  |- Average write latency : 0 cycle
|  |  |  |- No read activity
|  |  |  |- No write activity
|  |-[+] Node 1
|  |  |-[+] QoS 0
|  |  |  |- Minimum write latency : 5 cycles
```

```
|  |  |  |- Maximum write latency : 25 cycles
|  |  |  |- Average write latency : 6 cycles
|  |  |  |- 188 packets read
|  |  |  |- 184 packets written
|  |-[+] Node 2
|  |  |-[+] QoS 0
|  |  |  |- Minimum write latency : 6 cycles
|  |  |  |- Maximum write latency : 17 cycles
|  |  |  |- Average write latency : 7 cycles
|  |  |  |- 194 packets read
|  |  |  |- 214 packets written
|  |-[+] Node 3
|  |  |-[+] QoS 0
|  |  |  |- Minimum write latency : 7 cycles
|  |  |  |- Maximum write latency : 17 cycles
|  |  |  |- Average write latency : 9 cycles
|  |  |  |- 192 packets read
|  |  |  |- 205 packets written
|  |-[+] Node 4
|  |  |-[+] QoS 0
|  |  |  |- Minimum write latency : 6 cycles
|  |  |  |- Maximum write latency : 14 cycles
|  |  |  |- Average write latency : 8 cycles
|  |  |  |- 190 packets read
|  |  |  |- 197 packets written
|  |-[+] Node 5
|  |  |-[+] QoS 0
|  |  |  |- Minimum write latency : 5 cycles
|  |  |  |- Maximum write latency : 13 cycles
|  |  |  |- Average write latency : 5 cycles
|  |  |  |- 190 packets read
|  |  |  |- 200 packets written
```

Figure 4-2 : Aperçu des résultats de l'engin d’analyse lors de l'exécution du banc d’essai

En comparant les résultats du banc d'essai avec ceux obtenus par l'engin d'analyse, nous pouvons en déduire que les résultats concordent. Par le fait même, notre modèle haut niveau est validé. Par le fait même, nous validons que notre engin d'analyse est fiable.

4.1.2 Bas niveau

La validation du modèle bas niveau par un banc d'essai a été faite dans le cadre du projet de Jessica Allard-Bernier [22]. Par conséquent, l'explication exhaustive du banc d'essai ainsi que ces résultats ne sont pas présentés dans ce mémoire. Nous allons utiliser le modèle bas niveau résultant du processus de validation effectué dans le cadre de son travail. Nous considérons que le modèle bas niveau utilisé est validé et fonctionnel.

4.2 Charge maximale

Dans cette section, nous allons comparer la charge maximale (débit des messages) du modèle haut niveau avec le modèle bas niveau dans le but de s'assure que le modèle haut niveau est fiable. Nous allons accepter notre modèle haut niveau si le comportement de la charge maximale du modèle haut niveau est corrélé avec le comportement de la charge maximale du modèle bas niveau.

En rappel à la section 3.4.1.1, nous avons développé un modèle haut niveau se situant dans le niveau d'abstraction T-BCA « Transaction based Bus Cycle Accurate ». Un modèle T-BCA indique que les communications sont précises au cycle près tout en utilisant des interfaces. On ne s'attend pas à ce que la charge maximale du modèle haut niveau s'apparente complètement avec la charge maximale bas niveau, mais plutôt que l'allure globale soit respectée. Ceci signifie que les résultats peuvent différés, mais l'allure globale doit être similaire. Si tel est le cas, le modèle au niveau se situe dans le bon niveau d'abstraction. Dans le cas contraire, nous devrons réajuster le modèle haut niveau afin qu'il présente une précision acceptable.

Une charge maximale de 100 % signifie que les blocs fonctionnels branchés aux nœuds du réseau sur puce sont capables d'éjecter un paquet à chaque coup d'horloge. Puisqu'il existe

des latences de transmission, tel que présenté lors de la section 3.4.1.1, il est impossible d'obtenir un tel résultat. La charge maximale est obtenue par l'équation suivante, tirée de [34] :

$$TP = \frac{(Total\ messages\ completed)x\ (Message\ length)}{(Number\ of\ IP\ blocks)x\ (Total\ time)}$$

Afin de produire un graphique de la charge maximale, nous avons développé deux bancs d'essai (haut et bas niveau) constitués de 32 blocs matériels directement branchés sur le modèle du réseau sur puce (aucun adaptateur requis). Les deux bancs d'essai sont écrits dans deux langages différents, mais simulent le même comportement avec les mêmes délais. Les blocs matériels injectent un paquet à chaque coup d'horloge. Lorsqu'un paquet se présente à la sortie du réseau sur puce, le bloc matériel l'éjecte et en informe le tableau indicateur, du nom anglais *scoreboard*. L'objectif des bancs d'essai est d'éjecter un total de 320 000 paquets (environ 10,00 par nœuds). Lorsque l'objectif est atteint, on note le nombre de cycles requis.

4.2.1 Bas niveau

À l'aide du banc d'essaie bas niveau, nous avons obtenu les résultats présentés par le Tableau 1 avec lesquels nous avons obtenu la Figure 4-3. Sur cette figure, nous observons que la charge maximale augmente de façon linéaire (de 1 à 3 canaux) et qu'à un certain moment, la charge maximale est atteinte (à partir de 4 canaux).

Tableau 1 : Nombre de cycles requis pour l'éjection de 320 000 paquets à bas niveau

Nombre de canaux	Nombre de cycle requis	% de la charge maximale
1	82 741	12%
2	41 841	24%
3	28 547	35%
4	22 729	43%
5	21 713	46%
6	21 577	46%
7	21 499	47%
8	21 491	47%
10	21 491	47%
20	21 491	47%

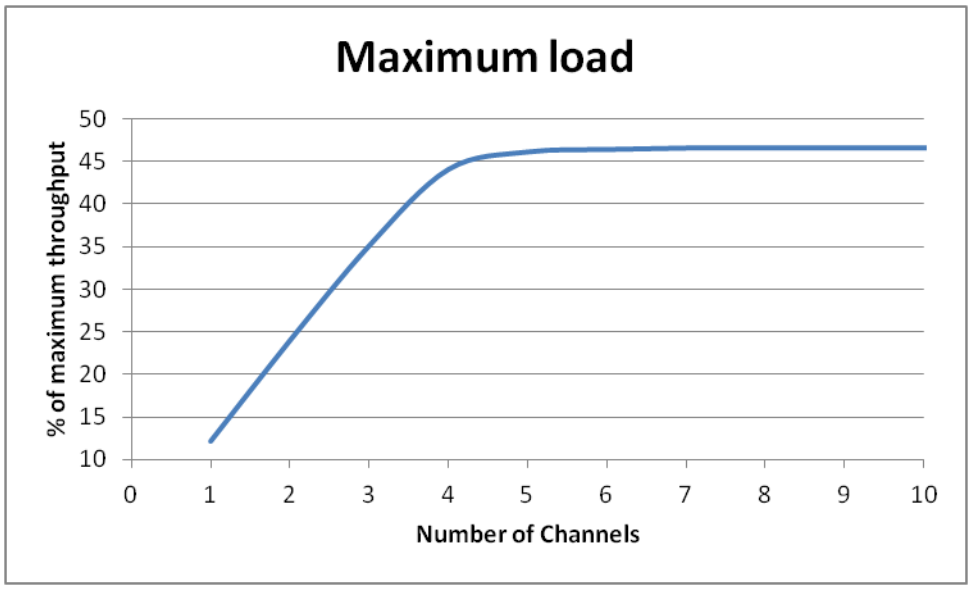

Figure 4-3 : Charge maximale du modèle bas niveau

4.2.2 Haut niveau

À l'aide du banc d'essaie haut niveau, nous avons obtenu les résultats présentés par le Tableau 2 avec lesquels nous avons obtenu la Figure 4-4. Sur cette figure, nous observons que la charge maximale augmente de façon linéaire (de 1 à 2 canaux) et qu'à un certain moment, la charge maximale est atteinte (à partir de 3 canaux).

Tableau 2 : Nombre de cycles requis pour l'éjection de 320 000 paquets à haut niveau

Nombre de canaux	Nombre de cycle requis	% de la charge maximale
1	41 942	24%
2	25 134	40%
3	23 433	43%
4	23 169	43%
5	23 127	43%
6	23 122	43%
7	23 100	43%
8	23 109	43%
10	23 109	43%
20	23 109	43%

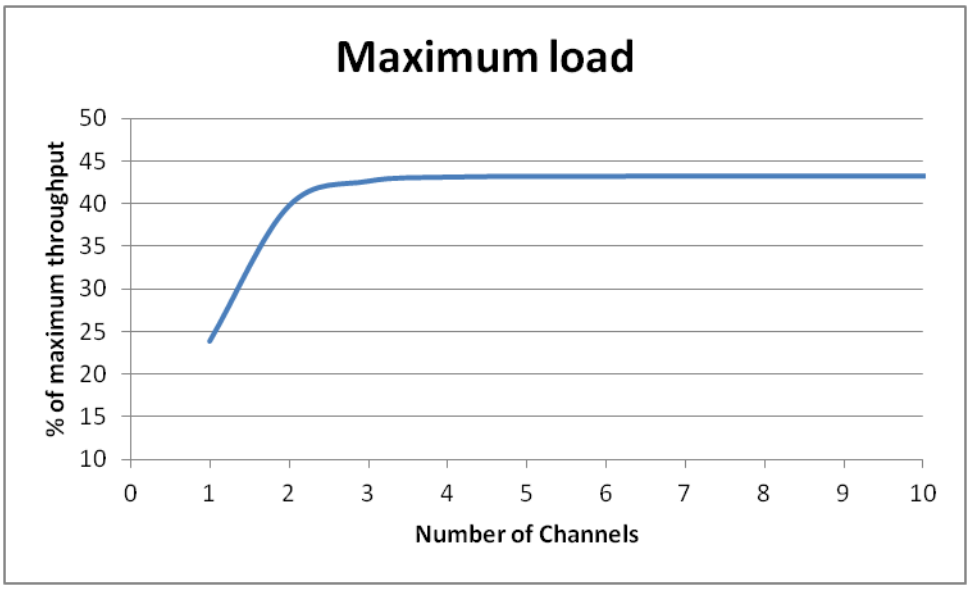

Figure 4-4: Charge maximale du modèle haut niveau

4.2.3 Comparaison

En superposant la Figure 4-3 avec Figure 4-4 , nous obtenons la Figure 4-5 sur laquelle il est possible de remarquer que la tendance des deux courbes ainsi que la charge maximale est semblable. En effet, la charge maximale obtenue pour le modèle haut niveau est de 43% tandis que la charge maximale obtenue pour le modèle bas niveau est de 47%. Il s'agit ici d'une erreur d'environ 4%.

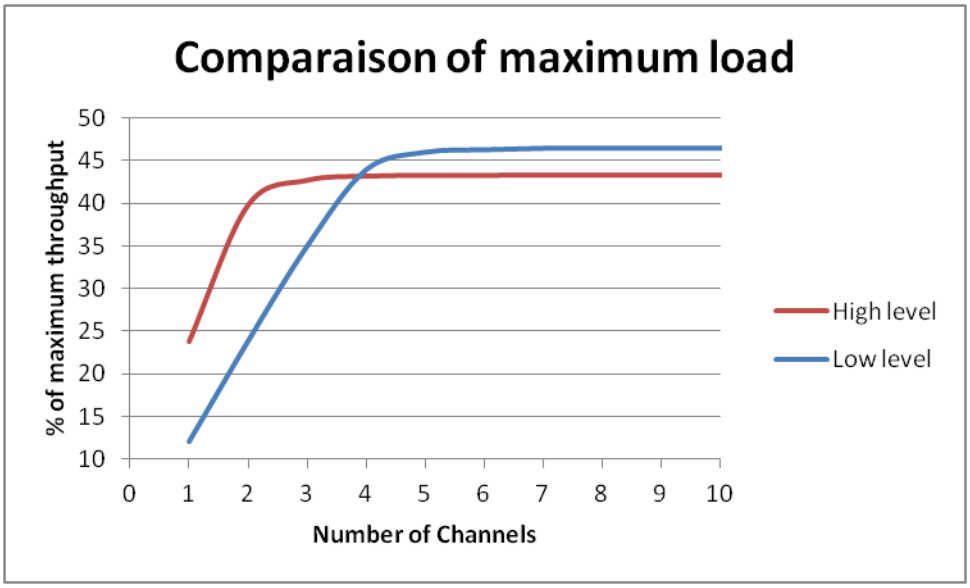

Figure 4-5 : Comparaison de la charge maximale

Tel que mentionné précédemment, la métrique permettant de quantifié la précision du notre modèle haut niveau est le facteur de corrélation (r_p) de la charge maximale des deux modèles. Le facteur de corrélation est une métrique permettant d'indiquer le degré de similitude entre deux séries de données. Le facteur de corrélation varie entre -1 et +1 où +1 indique une relation parfaite et -1, l'opposé. Notre seuil d'acceptation est l'atteinte d'un facteur de corrélation de 0.75. Le facteur de corrélation est obtenu par l'équation suivante :

$$r_p = \frac{\sum_{i=1}^{N}(x_i - \bar{x}) * (y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2} * \sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}}$$

où x représente les données de la charge maximale du modèle haut niveau et

y représente les données de la charge maximale du modèle bas niveau

À l'aide des données présentées par les tableaux Tableau 1 et Tableau 2, nous obtenons un facteur de corrélation de 0.88 ce qui dépasse largement notre seuil d'acceptation. Par conséquent, l'implémentation de notre modèle haut niveau semble très acceptable.

4.3 MJPEG

Dans cette section, nous allons nous intéresser à l'exécution de l'architecture présentée à la section 3.4.3. Nous présentons ici les résultats de simulation ainsi que l'exécution sur puce FPGA. Ces résultats démontrent le bon fonctionnement des modèles du Rotator-On-Chip utilisés dans une réelle application.

4.3.1 Simulation

Lors de la simulation de l'architecture de test dans l'outil de simulation SPACE, il est possible d'obtenir quatre résultats pertinents. Le premier résultat est celui de constater le bon fonctionnement du décodage de l'animation. Le deuxième est le temps réel de simulation communément appelé le « Wall clock ». Le troisième résultat est le temps de simulation (le temps simulé). Finalement, le dernier résultat présenté sera celui obtenu par l'engin d'analyse.

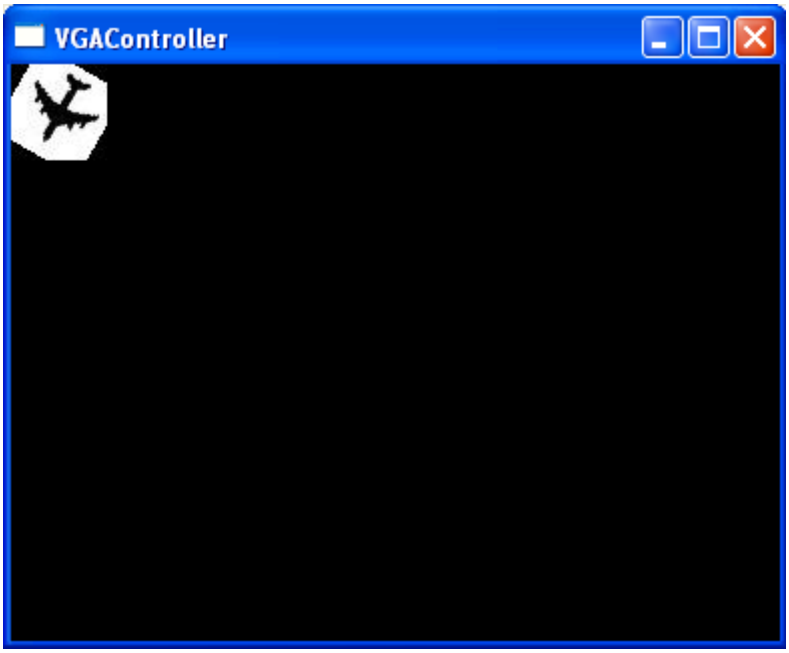

Figure 4-6 : Virtualisation VGA

Le bon fonctionnement de l'application se base sur la sortie vidéo du décodage de l'animation par l'entremise de la virtualisation VGA présentée lors de la section 3.4.4.1. La Figure 4-6 présente une image décodée lors de la simulation sur la plateforme virtuelle.

Tableau 3 : Résultats de simulation

Temps réel de simulation	24 minutes
Temps de simulation	2. 02 secondes

La Tableau 3 présente les résultats pour le temps réel de simulation ainsi que pour le temps de simulation. Il faut distinguer le tmeps réele de simulation et le temps de simulation. Le temps réel de simulation est en fonction de la machine hôte sur lequel la simulation roule. Dans notre cas, la machine hôte est un Intel Core 2 Duo cadencé à 3,16 GHz. À l'opposé, le temps de simulation (temps simulé) correspond au temps que devrait prendre l'exécution de l'architecture sur une puce FPGA ayant les mêmes caractéristiques (fréquence, architecture, précision des

modèles, etc.). La première constatation qu'il est qu'il faut attendre 24 minutes pour simuler 2. 02 secondes d'un décodage d'une animation MJPEG ayant 25 images. Ceci est uniquement vrai pour l'assignation multiprocesseur de la Figure 3-13. Vingt-quatre (24) minutes peuvent sembler interminables, mais il ne faut pas oublier qu'il y a cinq processeurs µBlaze à simuler via leur ISS. Par exemple, si l'on met tout en matériel, il y a beaucoup moins de composantes à simuler ce qui aura comme conséquence de diminuer le temps réel de simulation. Mentionnons également, que plus la machine hôte sera performante, plus ce chiffre aura tendance à diminuer.

Nous utilisons la référence de 30 images par secondes comme étant le temps réel. En effectuant une règle de trois avec le temps de simulation, l'application MJPEG telle que présentée par la Figure 3-13 effectue 10.54 images par seconde en simulation. Il faudrait donc un facteur d'accélération de 2.84 afin d'obtenir le temps réel.

Tableau 4 : Nombre de paquets entrant et sortant des nœuds

	Réception	Envoi
Nœud 0 (Demux)	0	27007
Nœud 4 (VLD)	26799	14400
Nœud 2 (IQZZ)	14601	28800
Nœud 1 (IDCT)	28801	7200
Nœud 3 (Libu)	7206	0

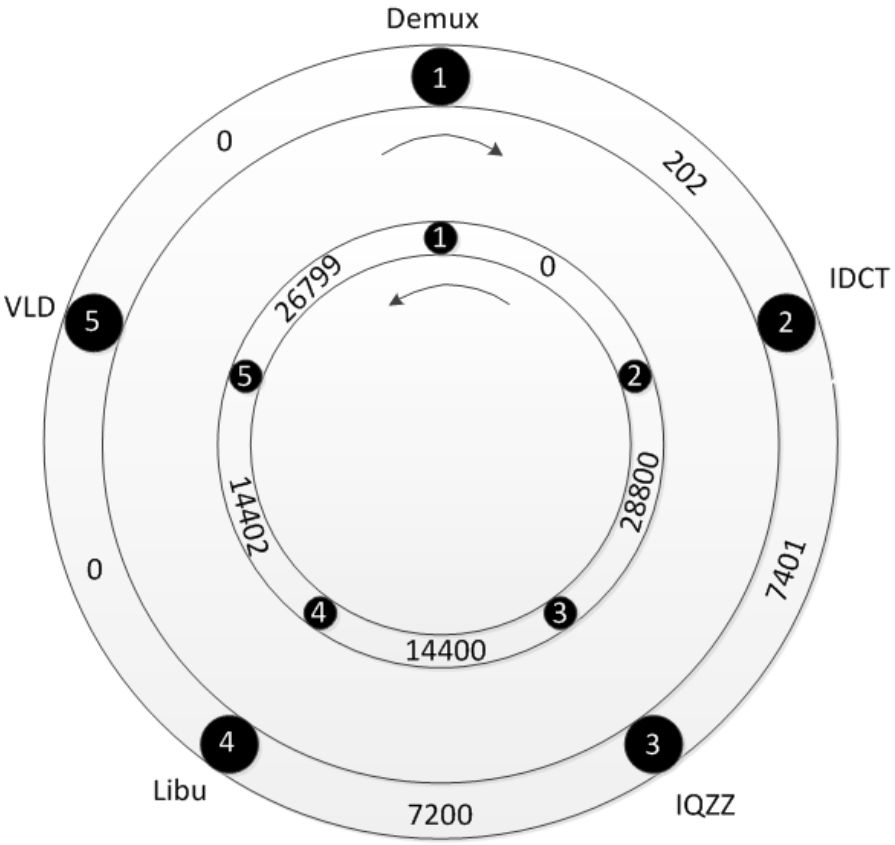

Figure 4-7 : Synthèse des communications

Afin d'obtenir le nombre de communications qui sont véhiculées sur le Rotator-On-Chip, nous nous rabattons sur l'engin d'analyse dûment validé. Ceci correspond à notre dernier résultat dans le cadre de la simulation de la plate-forme de test. Le Tableau 4 présentement le nombre de communications en termes de paquet entrant et sortant de chaque nœud tandis que la Figure 4-7 présente la synthèse des résultats obtenus par l'engin d'analyse. Sur cette figure, il est possible de voir le nombre de paquets qui circulent sur les liens de communications entre les différents nœuds.

Ces données sont utiles afin de comprendre le comportement de l'application dans le modèle du réseau sur puce. Par exemple, il est possible d'identifier les nœuds qui ont intérêt à être déplacés afin de diminuer la charge sur les liens de communication. En effet, il aurait été intéressant de placer les tâches de l'application MJPEG les unes à côté des autres autour du

Rotator-On-Chip selon la Figure 3-12. Ceci aurait eu comme effet de diminuer les communications pour ainsi tirer avantage du flot de données inhérent à l'application.

4.3.2 Raffinement

Lors de l'exécution du résultat de co-synthèse logicielle/matérielle sur la carte FPGA, il est plutôt difficile de récolter des résultats. Par conséquent, nous allons uniquement présenter deux résultats. Le premier résultat est la validation du bon fonctionnement de l'application de façon générale. Le dernier résultat est le temps requis pour l'exécution de l'application.

Figure 4-8 : MJPEG en action sur FPGA

Nous avons fait rouler l'application sur une puce Xilinx FPGA ML410. Le système ainsi synthétisé tourne à une fréquence de 100 MHz. À l'instar de la Figure 4-6, la Figure 4-8 présente une image décodée lors de l'exécution de la plate-forme de test sur la puce FPGA. Cette image provient d'un écran branché à la puce FPGA sur lequel tourne notre application ayant notre contrôleur VGA. Le restant des images décodées correspond à l'animation anticipée. Par conséquent, la validation du bon fonctionnement de l'application est validée par l'observation des 25 images affichées sur l'écran VGA par l'entremise de notre contrôleur VGA

Pour obtenir le temps d'exécution, nous avons utilisé une minuterie que l'on démarre au début de l'exécution et qui est arrêtée lorsque la dernière image est décodée. Grâce à cette minuterie, nous avons été en mesure d'observer qu'il faut 289 791 629 coups d'horloge pour parvenir à décoder les 25 images de l'animation. Ceci correspond à 2.89 secondes pour le décodage complet. En utilisant la règle de trois, l'application MJPEG exécutée sur puce FPGA effectue 8.65 images par seconde. Il faut donc une accélération de 3.46 afin d'obtenir le temps réel.

4.4 Discussion

La précision des modèles ralentit l'exécution de la simulation. En effet, plus les modèles sont précis (tend vers une représentation RTL), plus la simulation doit simuler le comportement bas niveau ce qui a pour effet d'augmenter le temps de simulation « Wall clock ». Bien que notre modèle du Rotator-On-Chip soit au niveau T-BCA, nous ne croyons pas que ça soit ce dernier qui ralentit la simulation. Nous croyons plutôt que la simulation est ralentie par les ISS simulés. En effet, le modèle du µBlaze tel que fournis par SPACE simule le « pipeline » ainsi que le détail fin de chaque instruction assembleur. Nous avons simulé une architecture semblable à la Figure 3-13 mais en remplaçant les ISS par des blocs matériels (donc une solution complètement matériel). Cette nouvelle architecture simulée nécessite moins d'une minute pour se terminer. Ceci confirme donc que la majeure partie du temps simulé est consumée par la simulation des ISS et non par notre modèle du Rotator-On-Chip.

Tableau 5: Sommaire des résultats

Simulation	10.54 images par seconde
Exécution sur puce FPGA	8.65 images par seconde
Erreur de simulation	18 %

En ce qui a trait aux résultats présentés, le Tableau 5 présente le sommaire des résultats obtenus. Il est intéressant de constater qu'en simulation, il est possible de décoder 10.54 images

par secondes tandis que lors de l'exécution sur puce FPGA, nous obtenons 8.65 images par secondes. Ceci correspond à une erreur de 18%. Étant donné les résultats obtenus à la section 4.2 sur le facteur de corrélation de la charge maximale, nous pouvons affirmer avec confiance qu'une grande partie de cette erreur ne provient pas du RoC, mais des disparités dans les différents modèles (ISS, FSL, etc.).

Comme nous l'avons dit dans l'introduction de la section courante, il est difficile d'obtenir des résultats lorsqu'on se trouve à bas niveau (complexité et granularité trop fine du langage). Ceci justifie l'utilité d'avoir un modèle haut niveau dans un outil haut niveau avec un engin d'analyse. Puisque notre modèle haut niveau est fiable, nous considérons que les résultats obtenus de l'engin d'analyse sont très voisins de ceux qu'on aurait obtenus si on avait fait le même exercice à bas niveau.

CHAPITRE 5 CONCLUSION

En conclusion, le recours aux réseaux sur puce est une réalité que nous ne pouvons nier. Les avantages que représente l'utilisation d'une telle topologie sont considérables. Le plus tôt qu'il est possible de simuler, avec un bon degré de précision, une architecture intégrant un réseau sur puce, le plus tôt il sera possible de prendre des décisions éclairées. Dans cette optique, l'objectif de ce mémoire est d'intégrer, simuler et valider une architecture utilisant un réseau sur puce dans un outil haut niveau.

Dans le cadre de ce mémoire, nous avons donc intégré un modèle de réseau sur puce dans un outil haut niveau. Le modèle intégré nommé Rotator-On-Chip permet d'étendre la librairie des topologies de communication et, par le fait même, de débloquer un nouveau pan dans l'exploration architecture. Par ailleurs, l'utilisation de l'engin d'analyse facilite l'extraction de métrique de performance pour les modèles de réseau sur puce.

Par conséquent, l'objectif de ce mémoire a été atteint dans la mesure où un modèle de réseau sur puce a été correctement intégré dans un outil haut niveau. Ce modèle a été simulé et validé à l'aide d'une plate-forme pour lesquelles nous avons présenté les résultats. Il a été possible de démontrer qu'une erreur de 18% existe entre la simulation et l'exécution sur puce FPGA d'une architecture test.

5.1 Travaux futurs

Afin de tirer pleinement profit d'un réseau sur puce, il serait intéressant d'utiliser une application robuste et de la paralléliser en ayant recours à un réseau sur puce. On pourrait ainsi comparer une telle approche à une approche utilisant une topologie de communication traditionnelle. Lors de cet exercice, il serait possible d'observer à quel moment l'utilisation d'un réseau sur puce pour une application donnée devient intéressante.

Dans l'optique de valider l'erreur observée, il faudrait faire d'autres tests avec d'autres systèmes afin de la valider. L'erreur peut être stable, moindre ou pire, elle peut s'accroître. Ces conclusions permettront de réajuster dans le cas échéant.

De plus, l'ajout de réseau sur puce d'architecture différente (architecture en arbre ou en maille) serait une prolongation logique de ce mémoire. Un tel ajout nécessiterait le développement d'un modèle haut niveau, d'un modèle bas niveau et finalement, le développement de l'adaptateur unitaire permettrait l'interopérabilité. Un tel ajout permettrait de comparer les caractéristiques et performances des réseaux sur puce

Finalement, la mise à jour des interfaces ainsi que du modèle haut niveau vers le nouveau standard TLM2 permettraient l'actualiser le modèle favorisant ainsi l'interopérabilité avec les blocs fonctionnels standard TLM2.

RÉFÉRENCES

[1] G. Moore, "Cramming more components onto integrated circuits," Electronics, vol. 38, no. 8, 1965.

[2] H. Yagi, E. Haritan, G. Smith, "ESL Hand-off: Fact or EDA Fiction?," *Design Automation Conference*, 2008, pp. 310-312.

[3] K. Hadjiat, F. St-Pierre, G. Bois et Y. Savaria, "An FPGA Implementation of a Scalable Network-on-Chip Based on the Token Ring Concept," *14th IEEE International Conference on Electronics, Circuits and Systems*, 2007, pp. 11-14.

[4] C. Smythe, "ISO 8802/5 token ring local-area networks," Electronics & Communication Engineering Journal, vol. 11, no. 4, 1999, pp. 195-207.

[5] D. Pham et al, "Overview of the Architecture, Circuit Design, and Physical Implementation of a First-Generation Cell Processor," *IEEE journal of solid-state circuits*, vol. 41, no. 1, 2006.

[6] F. Busaba, "VHDL description of self-checking logic circuits," *Proceedings of the Twenty-Eighth Southeastern Symposium on System Theory*, 1996, pp. 477-481.

[7] C. Dawson, S. Pattanam, D. Roberts, "The Verilog Procedural Interface for the Verilog Hardware Description Language," *IEEE International Verilog HDL Conference*, 1996, pp. 17-23.

[8] W. Ecker, V. Esen, M. Hull, "Impact of Description Language, Abstraction Layer, and Value Representation on Simulation Performance," *Design, Automation & Test in Europe Conference & Exhibition*, 2007, pp. 1-6.

[9] B. Bailey, G. Martin, A. Piziali, "ESL Design and Verification," États-Unis: Elsevier, 2006.

[10] E. Viaud, F. Pêcheux, "A New Paradigm and Associated Tools for TLM/TModeling of MPSoCs," Research in MIcroelectronics and Electronics, 2006, pp. 217-220.

[11] K. Man, "An overview of SystemC," *Research in Microelectronics and Electronics*, 2005, pp. 145-148.

[12] G. Oshiro, "*Network-on-chip refinement in a ESL tool*," M. Sc. A, École Polytechnique de Montréal, Montréal, Qc, Canada, 2011.

[13] D. Atienza et al, " Network-on-Chip design and synthesis outlook," *Integration, the VLSI Journal*, vol. 41, no. 3, pp 340-349, 2008.

[14] U. Ogras, J. Hu, R. Marculescu, "Key Research Problems in NoC Design: A Holistic Perspective," *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2005, pp. 69-74.

[15] T. Bjerregaard, S. Mahadevan, "A Survey of Research and Practices of Network-on-Chip," *ACM Computing Surveys*, vol. 38, no. 1, 2006.

[16] A. C. J Kienhuis, "Design Space Exploration of Stream based Data-flow Architectures," Ph. D. thesis, Delft University of Technology, 1998.

[17] G. Cyr, G. Bois et M. Aboulhamid, “Generation of processor interface for SoC using standard communication protocol,” *Proceedings Computers and Digital Techniques,* vol. 151, no. 5, pp 367-376, 2004.

[18] Chevalier. J et al, "A SystemC refinement methodology for embedded software," *Design & Test of Computers*, vol. 23, no. 2, 2006.

[19] ISO, "Open Systems Interconnection," ISO/IEC 7498-1, 1994, [En ligne]. Disponible: http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip. [Consulté le 16 septembre 2011].

[20] JPEG Committee, "Digital compression and coding of continuous-tone still images," ISO/IEC 10918-1, 1994, [En ligne]. Disponible: http://www.jpeg.org/. [Consulté le 16 septembre 2011].

[21] S. Kumar et al, "A network on chip architecture and design Methodology," *IEEE Computer Society Annual Symposium on VLSI*, 2002, pp. 117-124.

[22] J. Allard-Bernier, "*Méthode de reconfiguration dynamique pour un réseau sur puce tolérant aux fautes*," M. Sc. A, École Polytechnique de Montréal, Montréal, Qc, Canada, 2011.

[23] Mircium, "RTOS," *Micrium - RTOS*. [En ligne]. Disponible: http http://wwwmicrium.com/page/products/rtos. [Consulté le 30 juin 2010].

[24] J. Chevalier et al, "A SystemC Refinement Methodology for Embedded Software," *IEEE Design & Test of Computers*, 2006, pp 148-158.

[25] J. Chevalier et al, "SPACE: A Hardware/Software SystemC Modeling Plateform Including an RTOS," *Language for System Specification*, 2004, pp 91-104.

[26] SoCLib, "SoCLib," [En ligne], Disponible : http://www.soclib.fr. [Consulté le 30 juin 2011]

[27] O. Benny, "*Implémentation d’un modèle de communication transcationel dans une place-forme en SystemC*," M. Sc. A, École Polytechnique de Montréal, Montréal, Qc, Canada, 2003.

[28] L. Cai, D. Gajski, "Transaction Level Modeling: An Overview," *Hardware/Software Codesign and System Synthesis*, 2004, pp 19-24.

[29] J. A. Rowson, A. Sangiovanni-Vincentelli, "Interface-Based Design," *Proceedings of the 34th annual Design Automation Conference*, 2007, pp 178-183.

[30] S. Pasricha, N. Dutt, M. Ben-Romdhane, "Fast Exploration of Bus-Based Communication Architectures at the CCATB Abstraction," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 2, 2004.

[31] G. N.Khan, A. Tino, "Design and Simulation of Network on Chip Architectures and Systems," *Microsystems Research Laboratory Electrical and Computer Engineering*, 2011. [En ligne]. Disponible: http://www.cmoset.com/uploads/Gul_Khan_2010.pdf. [Consulté le 30 juin 2011].

[32] P. Guerrier, A. Greiner, "A Generic Architecture for On-Chip Packet-Switched Interconnections," *Proceedings of the Conference on Design and Test in Europe*, 2000, pp 250-256.

[33] F. Karim et al, "An Interconnect Architecture for Networking Systems on Chips," *IEEE Micro*, vol. 22, no. 5, pp 36-45, 2002.

[34] P. Pratim et al, "Performance Evaluation and Design Trade-Offs for Network-On-Chip Interconnect Architectures," *IEEE Transactions on Computers*, vol. 54, no. 8, pp 1025-1040, 2005.

[35] W.J. Dally, B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Network," *Proceedings Design Automation Conference*, 2001, pp 683-689.

[36] W.J. Dally, C.L. Seitz, "The Torus Routing Chip," California Institute of Technology, Pasaden, California, États-Unis, Technical Report 5208:TR: 86, 1986.

[37] P.P. Pande, C. Grecu, A. Ivanov et R. Saleh, "Design of a Switch for Network on Chip Applications," *Proceedings Circuits and Systems (ISCAS)*, vol.5, pp 217-220, 2003.

[38] D. Tutsch, M. Malek, "Comparison of Network-on-Chip Topologies for Multicore Systems Considering Multicast and Local Traffic," *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, 2009.

[39] B. Bailey, "Tool tackles hardware software codesign for ARM-based FPGAs," *EETimes*, 2011. [En ligne]. Disponible: www.eetimes.com/electronics-blogs/other/4230225/Tool-tackles-hardware-software-codesign-for-ARM-based-FPGAs. [Consulté le 13 novembre 2011].

[40] M. Langevin et C. Pilkington, "Hyper-ring-on-chip (HyRoC) architecture," *Brevet américain*, 7965725, 2006. [En ligne] http://www.uspto.gov/web/patents/patog/week25/OG/html/1367-3/US07965725-20110621.html

[41] Arteris, "Network on Chip (NoC) interconnect IP for System on Chip (SoC)," *Arteris*, 2011. [En ligne]. Disponible: www.arteris.com. [Consulté le 30 juin 2011].

[42] F. Deslauriers et al, "RoC: A Scalable Network on Chip Based on the Token Ring Concept," *IEEE North-East Workshop on Circuits and Systems*, 2006, pp. 157.

[43] D. Bertozzi, L. Benini, "Xpipes: A Network-on-Chip Architecture for gigascale Systems-on-Chip," *IEEE Circuits and Systems Magazine*, vol. 4, no. 2, pp. 18-31, 2004.

[44] J. Xu et al, "A Design Methodology for Application-Specific Networks-on-Chip," *ACM Transactions on Embedded Computing Systems*, vol. 5, no. 2, pp. 263-280, 2006.

[45] LIP6, "Laboratory of Computer Sciences, Paris 6," *University Pierre & Marie Curie*, 2011. [En ligne]. Disponible: http://www.lip6.fr/?LANG=en. [Consulté le 30 juin 2011].

[46] G. Bois, "Système embarqué," *Chapitre 1 : Introduction à la conception des systèmes embarqués*, 2011. [En ligne]. Disponible: www.cours.polymtl.ca/inf3610/Cours/Materiel/Chap1/chap1_intro_1pp.pdf. [Consulté le 30 juin 2011].

[47] OVP, "TLM 2.0," *OVP Technology enabled in OSCI TLM2.0*, 2011. [En ligne]. Disponible: http://www.ovpworld.org/technology_TLM2.0.php. [Consulté le 30 juin 2011].